Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

tesis fin de master

Máster Universitario en Inteligencia Artificial

Diseño y desarrollo de un modelo booleano probabilístico de regulación genética en el simulador de colonias bacterianas GRO

Autor: Irene Serrano Sebastián

Tutor: Alfonso Rodríguez-Patón Aradas

Julio, 2015

Departamento de Inteligencia Artificial

Resumen

Resulta interesante comprender como microorganismos sencillos como la bacteria *Escherichia coli* poseen mecanismos no tan simples para responder al entorno en el que se encuentran. Dicha respuesta está gestionada por complicadas redes de regulación formadas por genes y proteínas, donde cada elemento de la red genética debe tomar parte en armonía, en el momento justo y la cantidad adecuada para dar lugar a la respuesta celular apropiada.

La biología sintética es un nuevo área de la biología y la tecnología que fusiona la biología molecular, la ingeniería genética y las herramientas computacionales, para crear sistemas biológicos con funcionalidades novedosas. Los sistemas creados sintéticamente son ya una realidad, y cada vez se acumulan más trabajos alrededor del mundo que muestran su factibilidad. En este campo no sólo se hacen pequeñas modificaciones en la información genética, sino que también se diseñan, manipulan e introducen circuitos genéticos a los organismos.

Actualmente, se hace un gran esfuerzo para construir circuitos genéticos formados por numerosos genes y caracterizar la interacción de los mismos con otras moléculas, su regulación, expresión y funcionalidad en diferentes organismos. La mayoría de los proyectos de biología sintética que se han desarrollado hasta ahora, se basan en el conocimiento actual del funcionamiento de los organismos vivos. Sin embargo, la información es numerosa y creciente, por lo que se requiere de herramientas computacionales y matemáticas para integrar y hacer manejable esta gran cantidad de información.

El simulador de colonias bacterianas GRO posee la capacidad de representar las dinámicas más simples del comportamiento celular, tales como crecimiento, división y comu-

nicación intercelular mediante conjugación, pero carece de la capacidad de simular el comportamiento de la colonia en presencia de un circuito genético. Para ello, se ha creado un nuevo módulo de regulación genética que maneja las interaciones entre genes y proteínas de cada célula ejecutando respuestas celulares específicas.

Dado que en la mayoría de los experimentos intervienen colonias del orden de 10^5 individuos, es necesario un módulo de regulación genética simplificado que permita representar de la forma más precisa posible este proceso en colonias de tales magnitudes. El módulo genético integrado en GRO se basa en una red booleana, en la que un gen puede transitar entre dos estados, *on* (expresado) o *off* (reprimido), y cuya transición viene dada por una serie de reglas lógicas.

Abstract

It is interesting to understand how simple organisms such as *Escherichia coli* do not have simple mechanisms to respond to the environment in which they find themselves. This response is managed by complicated regulatory networks formed by genes and proteins, where each element of the genetic network should take part in harmony, at the right time and with the right amount to give rise to the appropriate cellular response.

Synthetic biology is a new area of biology and technology that combines molecular biology, genetic engineering and computational tools to create biological systems with novel features. The synthetically created systems are already a reality, and increasingly accumulate work around the world showing their feasibility. In this field not only minor changes are made in the genetic information but also genetic circuits designed, manipulated and introduced into the organisms.

Currently, it takes great effort to build genetic circuits formed by numerous genes and characterize their interaction with other molecules, their regulation, their expression and their function in different organisms. Most synthetic biology projects that have been developed so far are based on the current knowledge of the functioning of living organisms. However, there is a lot of information and it keeps accumulating, so it requires computational and mathematical tools to integrate and manage this wealth of information.

The bacterial colonies simulator, GRO, has the ability to represent the simplest dynamics of cell behavior, such as growth, division and intercellular communication by conjugation, but lacks the ability to simulate the behavior of the colony in the presence of a genetic circuit. To this end, a new genetic regulation module that handles inter-

actions between genes and proteins for each cell running specific cellular responses has been created.

Since most experiments involve colonies of about 10^5 individuals, a simplified genetic module which represent cell dynamics as accurately and simply as possible is needed. The integrated genetic GRO module is based on a Boolean network, in which a gene can be in either of two states, *on* (expressed) or *off* (repressed), and whose transition is given by a set of logical rules.

Índice general

Resumen **III**

Abstract **V**

Índice de figuras **IX**

Índice de tablas **XI**

1 Introducción **1**

1.1 Sistemas de Regulación Genética 1

1.2 Interacción Biología - Informática 2

1.3 Organización de la memoria 3

2 Fundamentos Biológicos **5**

2.1 Introducción 5

2.2 Ácido desoxirribonucléico (ADN) y Cromosomas 6

2.2.1 Estructura y función del ADN 7

2.2.2 Replicación del ADN 9

2.2.3 Síntesis de Proteínas 11

2.2.4 Virus y plásmidos 16

2.3 Expresión y regulación genética 19

2.3.1 Mecanismos de Control 19

2.3.2 Riboswitchs 20

2.3.3 Moléculas inductoras/inhibidoras 20

3 Circuitos Genéticos **23**
3.1 Introducción 23
3.2 Partes de los circuitos genéticos 25
3.2.1 Sensores 26
3.2.2 Circuitos 26
3.2.3 Actuadores 27
3.3 Circuitos básicos 28
3.3.1 Toggle Switch 28
3.3.2 Repressilator 29

4 Estado del Arte **31**
4.1 Simuladores de colonias bacterianas 31
4.1.1 CellModeller 32
4.1.2 iDynomics 33
4.1.3 BactoSIM 33
4.1.4 BacSIM 33
4.1.5 BSIM 34
4.1.6 DiSCUS 34
4.1.7 GRO 34
4.2 Herramientas para el diseño de circuitos genéticos 37
4.2.1 GenoCAD 37
4.2.2 EuGene 38
4.2.3 GEC 38
4.3 Modelos de Sistemas de Regulación Genética 39
4.3.1 Redes Booleanas 39
4.3.2 Ecuaciones Diferenciales y en diferencias 40

5 Solución Propuesta **43**
5.1 Modelo booleano probabilístico de regulación genética 44
5.1.1 Operones 45
5.1.2 Puertas Lógicas 45
5.1.3 Activaciones/degradaciones de proteínas y ARNs 48
5.1.4 Riboswitchs y moléculas inductoras 49
5.1.5 Estado inicial 50

5.1.6 Ruido 50
5.1.7 Actuadores 51
5.2 Diseño de circuitos genéticos 51
5.2.1 Parámetros 52
5.2.2 Matriz de ARNs-riboswitchs 52
5.2.3 Matriz de moléculas inductoras 53
5.2.4 Operones 54
5.2.5 Tiempos de degradación 57
5.2.6 Matriz de plásmidos 58
5.2.7 Acciones 59
5.2.8 E. Coli 60
5.2.9 Agregar molécula inductora 60
5.3 Procedimiento 61

6 Resultados y Validación **67**
6.1 Repressilator 68
6.2 Ruido 75
6.3 Supervivencia de plásmidos 78

7 Conclusión **83**

A Diseño general de un circuito genético en *gro* **89**

B Diseño del Repressilator en *gro* **91**

C Test de ruido en *gro* **94**

D Survival en *gro* **96**

Índice de figuras

2.1 Estructura de las bases purinas [33] 8
2.2 Estructura de doble hélice del ADN [35] 9
2.3 Replicación del ADN [35] 10
2.4 Proceso de transcripción [35] 13
2.5 Síntesis de proteínas [35] 14
2.6 Estructura de las proteínas [35] 15
2.7 Plásmido [36] 17
2.8 Proceso de conjugación [36] 18
2.9 Riboswitch [15] 20

3.1 Comparación Biología-Informática [3] 25
3.2 Diseño de una puerta NAND conectada a un inversor [44] 27
3.3 Circuito Toggle Switch [17] 28
3.4 Circuito Repressilator [16] 29

4.1 GRO 35
4.2 Ejemplo de red booleana síncrona [35] 40

5.1 Simplificación de concentración de proteína 48
5.2 Lista de plásmidos contenidos en una célula 51
5.3 Tiempo de activación/degradación en inducción 63

6.1 Circuito genético del Repressilator 68
6.2 Repressilator modelo 69
6.3 Expresión de la GFP en una célula real en la que se ha introducido el circuito Repressilator [16] 70

6.4 Estado booleano de la proteína GFP en una simulación del *Repressilator* 70
6.5 Concentración de GFP en una simulación del *Repressilator* 71
6.6 Concentración media de GFP obtenida en 10 simulaciones del *Repressilator* 72
6.7 Estado booleano de la proteína TetR en una simulación del *Repressilator* 72
6.8 Estado booleano de la proteína LacI en una simulación del *Repressilator* 73
6.9 Estado booleano de la proteína cI en una simulación del *Repressilator* . 73
6.10 Estados booleanos de las proteínas TetR, LacI y cI en una simulación del *Repressilator* 74
6.11 Cantidad de células que tienen cada una de la proteínas en estado *on* en una simulación del *Repressilator* 75
6.12 Ruido en una colonia 76
6.13 Diseño del circuito survival 79
6.14 Ejemplo de simulación survival 79

Índice de tablas

5.1 Puerta YES con activador. 46
5.2 Puerta YES con represor. 46
5.3 Puerta AND con dos activadores. 46
5.4 Puerta AND con dos represores. 46
5.5 Puerta AND con activador y un represor. 46
5.6 Puerta OR con dos activadores. 47
5.7 Puerta OR con dos represores. 47
5.8 Puerta OR con un activador y un represor. 47
5.9 Matriz de RNAs que activan riboswitchs 53
5.10 Matriz de moléculas que inducen/inhiben alguna proteína 54
5.11 Matriz de plásmidos 58

Capítulo 1

Introducción

1.1 Sistemas de Regulación Genética 1
1.2 Interacción Biología - Informática 2
1.3 Organización de la memoria 3

1.1. Sistemas de Regulación Genética

El objetivo principal de la biología sintética es crear módulos biológicos capaces de realizar diferentes funciones básicas, tales como sensores para recibir señales ambientales, actuadores para realizar reacciones fisiológicas, y procesadores para manipular el flujo de información celular [18][31][27][2]. Entre estas funciones, el procesamiento de información es esencial para integrar y almacenar las entradas ambientales y producir las salidas adecuadas tras el cómputo *in vivo*. Amplios estudios sobre el procesamiento de información de las células se han realizado a nivel de transcripción y traducción [20][29][40] facilitando el diseño de circuitos biológicos artificiales.

Se han desarrollado muchos modelos matemáticos para los sistemas de regulación genética [38][39][21]. Un enfoque común es describir los cambios de niveles de expresión utilizando ecuaciones diferenciales ordinarias (EDOs). De esta manera, una red se describe como un sistema dinámico comprendido por un conjunto de EDOs enlazadas. El uso de EDOs permite conseguir una mejor aproximación a los resultados experimentales.

Sin embargo, por lo general, no es posible determinar soluciones analíticas y el análisis numérico puede ser computacionalmente costoso. Además, las EDOs no describen los niveles de expresión de un individuo, modelan la regulación de genes de la colonia completa. Por estas razones, se han desplegado métodos con aspectos booleanos. Con este marco, los eventos regulatorios se representan como interruptores on-off.

Representar sistemas de regulación genética en colonias formadas por miles de células con EDOs puede ser un problema complejo debido a los problemas computacionales. Por esa razón, en este proyecto se ha diseñado un modelo booleano de regulación genética guiado por probabilidades de éxito en los procesos que intervienen, como la adherencia de un factor de transcripción al promotor o la traducción de ADN en ARN y la transcripción de ARN en proteína. Este modelo simplificado permite que cada individuo posea su propio sistema de regulación genética y actúe de forma independiente. Consiguiendo así, la capacidad de simular el comportamiento de circuitos genéticos en una colonia regidos por las dinámicas individuales de expresión genética.

1.2. Interacción Biología - Informática

Este campo de la Inteligencia Artificial necesita de una interacción constante entre biólogos, informáticos y matemáticos. El modelado de las dinámicas celulares requiere un conocimiento biológico que describa los procesos a representar, una visión matemática que traduzca esos procesos a ecuaciones y reglas lógicas y una visión informática que defina restricciones computacionales e implemente el modelo.

Esta interación es bidireccional, ya que los biólogos nutren a los informáticos con la descripción y uso de las herramientas biológicas empleadas, y los informáticos proveen de modelos, análisis y datos a los biólogos para guiar sus experimentos. Una vez implementado el modelo, los biólogos estudian los datos obtenidos y proponen modificaciones en caso de que fuera necesario. Las cuales deben de volverse a diseñar e implementar para obtener nuevos resultados. Este ciclo se realiza tantas veces como sea necesario hasta que se obtengan resultados acorde con los experimentos reales.

1.3. Organización de la memoria

La biología sintética hace confluir campos como la biología molecular, la genética y la biología de sistemas. Por otro lado, los modelos matemáticos y las ciencias computacionales son necesarias para poder simular circuitos genéticos. Esto convierte a la biología sintética en un campo que requiere una preparación específica en diversos ámbitos de la ciencia. Con el fin de facilitar la comprensión de las diferentes perspectivas de la biología sintética, son necesarios algunos capítulos introductorios a estas ciencias. Por ello, la memoria se organiza como sigue:

Capítulo 1 Introducción de la memoria

Capítulo 2 Estudio de las principales características de la Expresión Genética. Para poder entender este proceso es necesaria una breve introducción al ADN y los procesos a seguir para sintetizar proteínas.

Capítulo 3 Estudio de los circuitos genéticos, así como los dos circuitos más conocidos usados en biología sintética.

Capítulo 4 Estudio de los principales simuladores de colonias bacterianas existentes, así como de las herramientas de diseño de circuitos genéticos y otros modelos utilizados para representar la regulación genética.

Capítulo 5 Descripción del modelo implementado en el simulador GRO para representar la expresión genética, además de las nuevas funciones creadas en lenguaje *gro* para el diseño de circuitos genéticos.

Capítulo 6 Resultados obtenidos al recrear el circuito *Repressilator*, además de la validación del funcionamiento del ruido y una posible aplicación del modelo.

Capítulo 7 Conclusiones

Bibliografía Material bibliográfico utilizado.

Anexos Diseños en lenguaje *gro* de los circuitos mencionados en esta memoria.

Capítulo 2

Fundamentos Biológicos

2.1 Introducción 5
2.2 Ácido desoxirribonucléico (ADN) y Cromosomas 6
2.2.1 Estructura y función del ADN 7
2.2.2 Replicación del ADN 9
2.2.3 Síntesis de Proteínas 11
2.2.4 Virus y plásmidos 16
2.3 Expresión y regulación genética 19
2.3.1 Mecanismos de Control 19
2.3.2 Riboswitchs 20
2.3.3 Moléculas inductoras/inhibidoras 20

2.1. Introducción

Casi todas las características microbianas son controladas o influidas por la herencia. Las características microbianas heredadas comprenden su forma y aspecto estructural, su metabolismo, su capacidad de moverse o comportarse de diversas maneras y su capacidad de interactuar con otros organismos, quizá para causar una enfermedad. Los microorganismos individuales transmiten esta característica a su descendencia a través de los genes.

La información genética contenida en una célula se denomina genoma. El genoma de una célula incluye sus cromosomas y plásmidos. Los cromosomas y los plásmidos son estructuras que contienen ADN y que fisicamente portan información hereditaria; Los cromosomas y los plásmidos contienen los genes. Los genes son segmentos de ADN (salvo en algunos virus, los cuales están formados por ARN) que codifican productos funcionales.

El genotipo de un organismo es su constitución genética, la información que codifica la totalidad de la características particulares del organismo. El genotipo representa propiedades potenciales pero no las propiedades mismas. El fenotipo se refiere a las propiedades reales y expresadas, como la capacidad del organismo de llevar a cabo una reacción química particular. Por lo tanto, el fenotipo es la manifestación del genotipo.

En términos moleculares el genotipo de un organismo es su colección de genes, su ADN completo, y su fenotipo es su colección de proteínas. La mayoría de las propiedades de una célula derivan de las estructuras y las funciones de sus proteínas. Por ejemplo, la presencia de la proteína GFP hace que el microorganismo emita una fluorescencia verde.

2.2. Ácido desoxirribonucléico (ADN) y Cromosomas

La información genética reside en la molécula de ADN; en ella está codificado cómo, cuándo, y dónde se han de producir los procesos de crecimiento, desarrollo, diferenciación y regulación que de las células que configuran los organismos vivos. El conjunto completo de información genética en un ser vivo se denomina genoma. El genoma humano está formado por hebras de ADN enrolladas y proteínas asociadas, organizadas en densas estructuras denominadas cromosomas. Más allá de las diferencias fisiológicas, según como distribuyen su material genético se distingue entre células eucariotas y procariotas.

En las células eucariotas, de las que están compuestos los animales, el material genético común de todas las células se encuentra en un orgánulo aislado por una membrana denominado núcleo. Las células procariotas, las bacterias, no disponen de membrana nuclear que separe el núcleo del resto de componentes. Dado que este documento gira en torno a un simulador de colonias bacterianas, a partir de este momento la memoria se enfocará en las células procariotas.

2.2.1. Estructura y función del ADN

El ADN es una gran molécula formada por dos hebras enlazadas que describen una hélice doble que gira en torno a un eje central. Cada hebra esta formada por elementos más simples denominados nucleótidos. Los nucleótidos a su vez están formados por bases aromáticas (purinas o pirimidinas), azúcares y grupos fosfatos. Las dos hebras se mantienen unidas gracias al apareamiento o atracción fisico-química establecida entre las bases de los nucleótidos de ambas hebras.

2.2.1.1. Componentes del ADN

Las bases del ADN son compuestos cíclicos que contienen nitrógeno. En el ADN están presentes las bases adenina (A), guanina (G), timina (T) y citosina (C). En el ARN no aparece la timina que es sustituida por el uracilo (U). Según su estructura se clasifican como purinas o pirimidinas. En la figura 2.1, las bases purinas son las de la parte superior (Adenina y Guanina) y las bases pirimidinas las de la parte inferior (Timina, Citosina y Uracilo).

Los nucleótidos (adenosina, guanosina, timidina y citidina) son los términos dados a la combinación de base y azúcar. El enlace entre la base y el azúcar se denomina glucosídico. El término nucleótido se refiere a la unión base + azúcar + grupo fosfato. Estos nucleótidos son elementos básicos de construcción de los ácidos nucléicos, ADN y ARN. Estos ácidos son largos polímeros formados por nucleótidos unidos covalentemente gracias a la formación de un enlace entre el grupo hidroxilo 3' del residuo de azúcar de un nucleótido y el grupo fosfato 5' del nucleótido siguiente. Este enlace se forma durante la síntesis bioquímica del ADN gracias a la enzima ADN polimerasa. Este enlace también está presente en el ARN aunque es catalizado por una enzima di-

ferente, la ARN polimerasa. Es importante destacar que los polinucleótidos tienen dos extremos diferentes denominados 5' y 3'. Frecuentemente el ADN posee con un grupo hidroxilo en el extremo 3' y un grupo fosfato en el extremo 5'. La lectura de las bases que forman parte de un ácido nucleico se realiza en sentido 5'→3'.

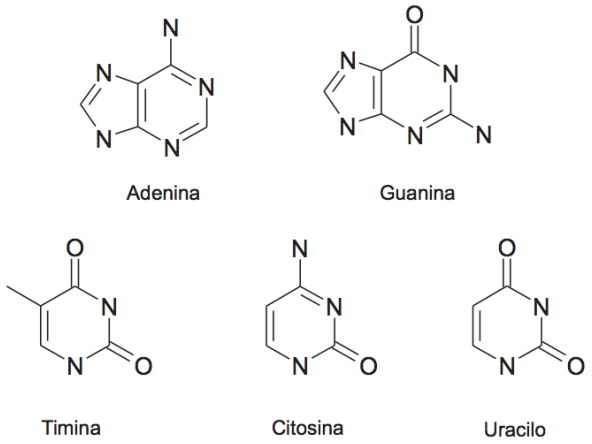


Figura 2.1: Estructura de las bases purinas [33]

2.2.1.2. Estructura de doble hélice del ADN

El ADN está formado por dos hebras o cadenas de ADN individuales alineadas de una forma antiparalela. Esto significa que una hebra está orientada en la dirección 5'→3' y la otra en dirección 3'→5'. Las dos hebras se mantienen juntas a través de enlaces de hidrógeno entre las bases. Las bases poseen la capacidad de unirse unas a otras de forma específica. Cada base se aparea siempre con su complementaria. Una base purina se une con una pirimidina; en concreto la adenina solo se aparea con la timina y la guanina con la citosina. El par de bases A·T está unido por dos enlaces de hidrógeno mientras que entre las bases C·G se establece una unión más fuerte de tres enlaces de hidrógeno.

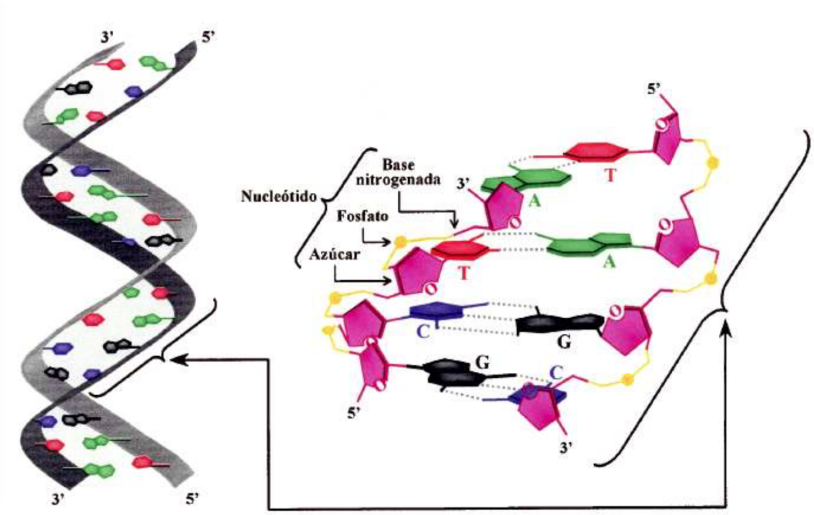


Figura 2.2: Estructura de doble hélice del ADN [35]

2.2.1.3. Función del ADN

La secuencia de bases de las hebras de ADN contienen la información necesaria para la vida de una célula. Esta molécula está diseñada para proteger la información genética por dos razones. Las bases portadoras de la información genética están en el interior de la doble hélice protegidas por el rígido esqueleto externo de azúcar-fosfato. Por otro lado, la presencia de dos hebras complementarias ofrece la posibilidad de disponer de dos patrones idénticos para la replicación durante la división celular. Esto permite que, en caso de dañarse una de las dos hebras, la complementaria servirá de plantilla para reparar la hebra dañada. El apareamiento de las bases y la complementariedad de las hebras son la base física molecular de la herencia y la evolución.

2.2.2. Replicación del ADN

La replicación del ADN posibilita el flujo de la información genética de una generación a la siguiente. Como se muestra en la figura 2.3, el ADN de una célula se replica antes de la división celular de modo que cada célula de la descendencia recibe un cromosoma idéntico al de la célula progenitora.

En la replicación del ADN, una molécula de ADN “parental” de doble cadena se convier-

te en dos moléculas "hijas" idénticas. Dado que las bases a lo largo de las dos cadenas de la doble hélice del ADN son complementarias, una cadena puede actuar como molde para la formación de la otra cadena.

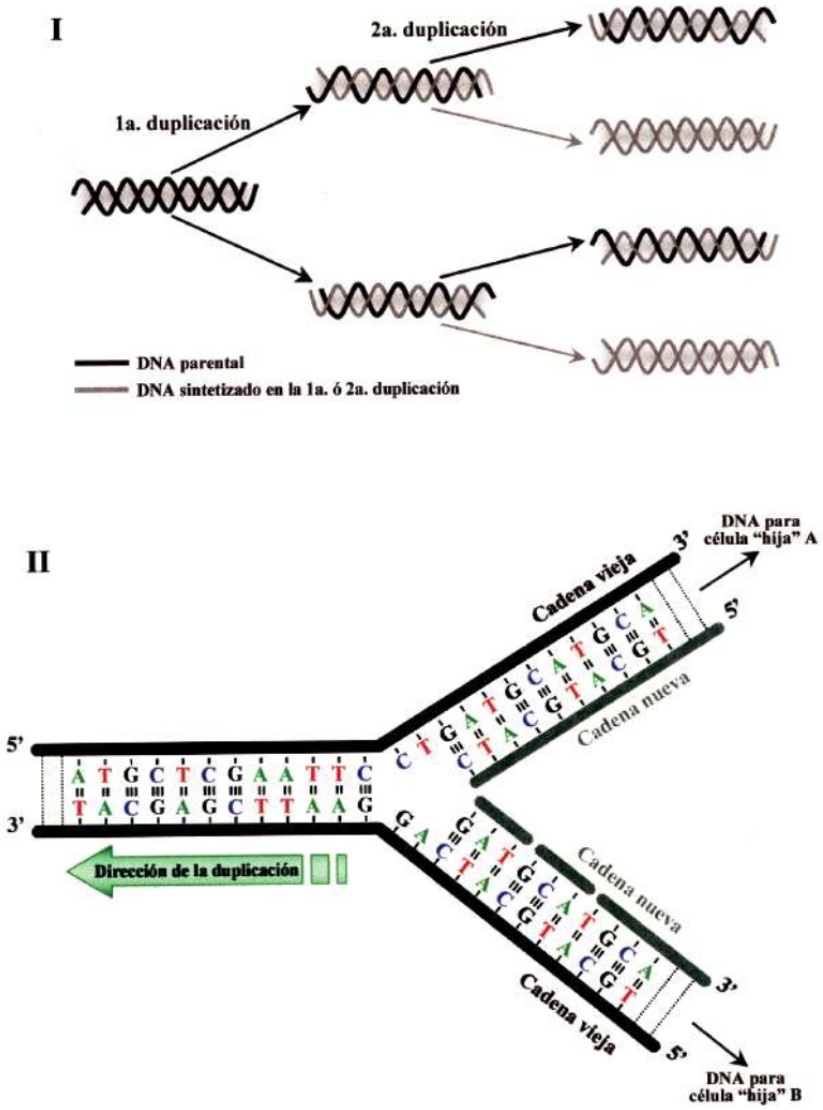


Figura 2.3: Replicación del ADN [35]

Cuando comienza la replicación, las dos cadenas de ADN parental son desenrolladas y se separan sucesivamente en pequeños segmentos de ADN. Los nucleótidos libres pre-

sentes en el citoplasma de la célula se aparean con las bases expuestas de ADN. Cuando hay timina (T) en la cadena original sólo puede unirse la adenina (A) a la nueva cadena; cuando hay guanina (G) en la cadena original sólo puede unirse la citosina (C) y así sucesivamente. Una vez alineados los nucleótidos, se unen a la cadena de ADN en crecimiento mediante una enzima denominada ADN polimerasa. Entonces el ADN parental se desenrolla un poco más para permitir el agregado de los nucleótidos siguientes. El punto en el cual se produce la replicación se denomina horquilla de replicación. A medida que la horquilla de replicación se desplaza a lo largo del ADN parental, cada una de las cadenas desenrolladas y separadas se combina con nuevos nucleótidos. La cadena original y la cadena hija recién sintetizada se vuelven a enrollar.

La orientación de las cadenas de ADN afecta en la replicación dado que las ADN polimerasas sólo pueden agregar nuevos nucleótidos al extremo 3'. Por consiguiente, dado que dos cadenas se tienen que unir en direcciones contrarias, las dos cadenas nuevas deben crecer en direcciones diferentes.

2.2.3. Síntesis de Proteínas

En el proceso de transcripción la información genética se copia (o se transcribe) en una secuencia de bases complementaria del ARN. Luego, la información codificada en este ARN se sintetiza en proteínas específicas a través del proceso de traducción.

2.2.3.1. Transcripción

La transcripción es la síntesis de una cadena de ARN a partir de un molde de ADN. En las células bacterianas hay tres clases de ARN: ARN mensajero, ARN ribosómico y ARN de transferencia. El ARN ribosómico forma parte de los ribosomas, la maquinaria celular para la síntesis de proteínas, el ARN de transferencia forma parte principalmente del proceso de traducción, por lo tanto se analizará en el siguiente apartado. Y, por último, el ARN mensajero (mRNA) transporta la información genética presente en los genes hasta los ribosomas, en el citoplasma, donde se realiza la traducción de esa información a proteína.

Durante la transcripción se sintetiza una cadena de mRNA con un gen específico como molde. En otras palabras, la información genética almacenada en el ADN se vuelve a escribir para que en la secuencia de bases del mRNA aparezca la misma información. Como en la realización del ADN, una G en el molde de ADN determina una C en el mRNA que se está formando, una C en el molde de ADN determina una G en el mRNA y una T en el molde de DNA determina una A en el mRNA. Sin embargo, una A en el molde de ADN determina un uracilo (U) en el mRNA porque el ARN contiene U en lugar de T. El Uracilo tiene una estructura química algo diferente de la T, pero se aparea de la misma manera. Por ejemplo, si el fragmento del molde de ADN tiene la secuencia de bases 3'-ATGCAT, la cadena de mRNA recién sintetizada tendrá la secuencia de bases complementaria 5'-UACGUA.

El proceso de transcripción requiere una enzima denominada ARN polimerasa y un conjunto de nucleótidos de ARN (fig. 2.4). La ARN polimerasa se une al ADN en un sitio denominado promotor. Sólo una de las dos cadenas de ADN actúa como molde para la síntesis de ARN de un gen concreto. Como sucede con el ADN, el ARN se sintetiza en la dirección 5'→3'. A continuación, la ARN polimerasa ensambla nucleótidos libres en una cadena nueva, para lo cual utiliza el apareamiento de bases complementarias como guía. A medida que la nueva cadena de ARN crece, la ARN polimerasa se desplaza a lo largo del ADN. La síntesis de ARN continúa hasta que la ARN polimerasa alcanza un sitio sobre el ADN denominado terminador (secuencia de terminación). Cuando esto sucede, la ARN polimerasa y el mRNA recién formado se liberan del ADN.

El proceso de transcripción permite que en un breve periodo la célula produzca copias de genes que pueden utilizarse como fuente directa de información para la síntesis de proteínas. El ARN mensajero actúa como intermediario entre la forma de almacenamiento permanente, el ADN y el proceso que utiliza la información, la traducción.

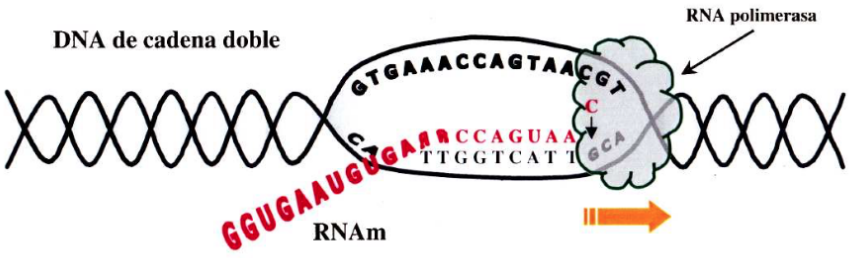


Figura 2.4: Proceso de transcripción [35]

2.2.3.2. Traducción

La traducción comprende la descodificación del "lenguaje" de los ácidos nucléicos y la conversión de esa información en el "lenguaje" de las proteínas.

El lenguaje del mRNA está en forma de codones, grupos de tres nucleótidos, como AUG, GGC o AAA. La secuencia de codones en una molécula de mRNA determina la secuencia de aminoácidos que formarán parte de la proteína que se sintetice. Cada codón codifica un aminoácido particular.

Los codones de mRNA se convierten en una proteína a través del proceso de traducción. Los codones de un mRNA se "leen" de modo secuencial y, en respuesta a cada codón, se ensambla el aminoácido apropiado en una cadena en crecimiento. El sitio de la traducción es el ribosoma y las moléculas de ARN de transferencia (tRNA) reconocen los codones específicos y transportan los aminoácidos necesarios.

Cada molécula de tRNA tiene un anticodón, una secuencia de bases complementarias de un codón. De esta manera, una molécula de tRNA puede aparear las bases con su codón asociado. Cada tRNA también puede transportar en su otro extremo el aminoácido codificado por el codón que reconoce el tRNA. Las funciones del ribosoma son dirigir de modo ordenado la unión de los tRNA a los codones y ensamblar los aminoácidos en la cadena para producir por último la proteína.

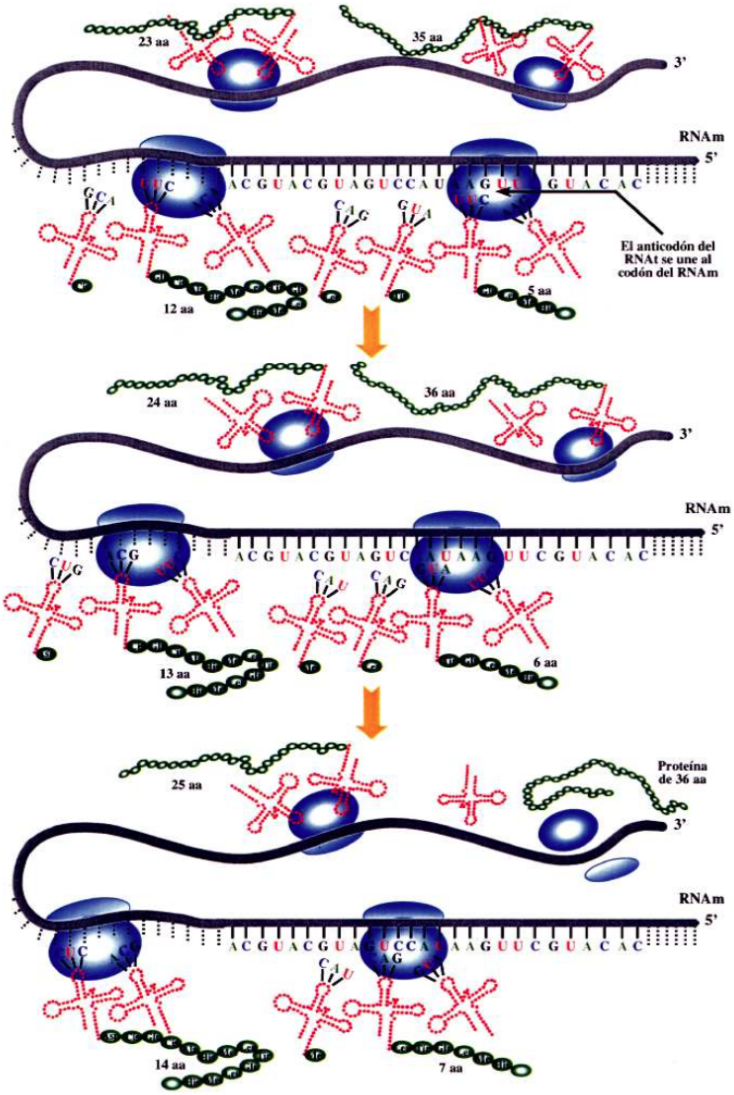


Figura 2.5: Síntesis de proteínas [35]

Los componentes necesarios para ensamblar son: las dos subunidades ribosómicas, un tRNA y la molécula de mRNA que se va a traducir. Esto sitúa al codón iniciador en la posición apropiada para que comience la traducción. El primer tRNA se une al codón iniciador, que lleva consigo el aminoácido metionina. Cuando el tRNA que reconoce el segundo codón se acerca a la posición sobre el ribosoma, este transfiere el primer aminoácido. Después, el ribosoma une los dos aminoácidos con un enlace peptídico y la molécula de tRNA lo abandona. Entonces el ribosoma se desplaza a lo largo del mRNA

hasta el codón siguiente. De esta forma, se van formando enlaces peptídicos en línea entre los aminoácidos consiguiendo una cadena polipéptídica. La traducción termina cuando el ribosoma llega a uno de los tres codones de terminación del mRNA. En este momento se liberan el mRNA y la cadena de aminoácidos recién sintetizada. Entonces el mRNA y los tRNA están disponibles para ser utilizados de nuevo (fig. 2.5).

2.2.3.3. Proteínas

Como ya se ha comentado, las proteínas son polímeros formados por decenas o centenas de aminoácidos que pueden ser combinados de entre veinte tipos diferentes. Cada proteína tiene una secuencia específica de aminoácidos de acuerdo a la secuencia de codones del gen que la codifica. La molécula resultante se conoce como estructura primaria de la proteína. A partir de esta secuencia primaria, la proteína puede adoptar una estructura secundaria que puede ser fundamentalmente de dos tipos: hélice o plegada. Las estructuras secundarias, a su vez, permiten el doblamiento de las proteínas en estructuras terciarias. Finalmente, las estructuras terciarias permiten la asociación de varias moléculas de proteínas en lo que se conoce como estructura cuaternaria (fig. 2.8).

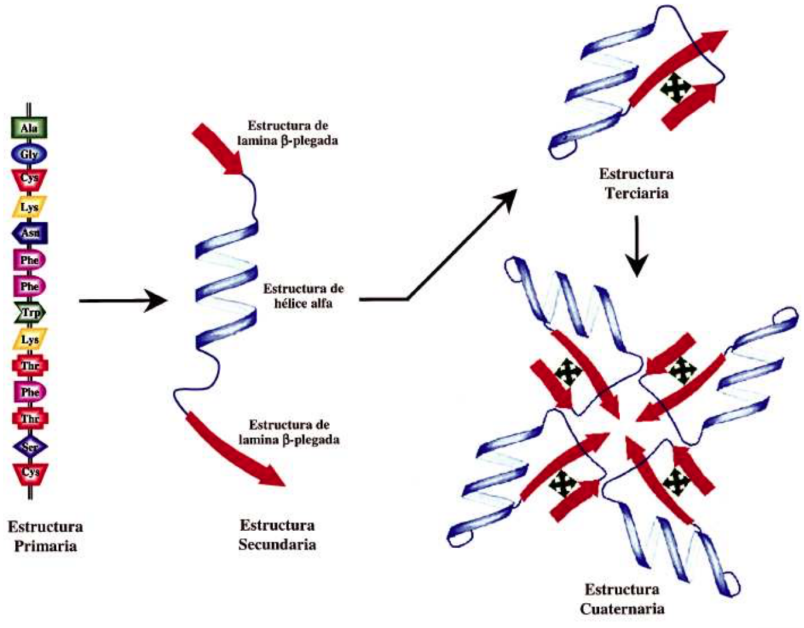


Figura 2.6: Estructura de las proteínas [35]

2.2.4. Virus y plásmidos

Estos elementos son secuencias de ADN que pueden replicarse de forma independiente del genoma de sus huéspedes. De ellos, los virus son los más independientes ya que poseen un revestimiento proteico que les permite moverse de una célula a otra. Los plásmidos (moléculas circulares de ADN) han de replicarse dentro de una única célula.

2.2.4.1. Virus

Cuando un virus penetra en una célula, su ADN se libera de la envoltura proteica (cápside) que lo protege. El ADN vírico, insertado y camuflado en el ADN del huésped, se transcribe en ARN y se traduce en las proteínas de su envoltura. El ADN vírico también se replica originando dentro de su huésped numerosas copias. A continuación, se ensambla el ADN vírico con su envoltura proteica y salen numerosos virus de la célula infectada que ocasionalmente es destruida.

Hay virus que sólo poseen ARN en lugar de ADN. Son los denominados retrovirus, que disponen de una enzima especial, la transcriptasa inversa. Esta enzima es una ADN polimerasa poco habitual que puede utilizar como patrón tanto ARN como ADN. Esta enzima del retrovirus, al entrar en la célula, hace una copia en ADN de su cadena de ARN dando lugar a una cadena híbrida ADN-ARN que es utilizada por la misma enzima para generar una doble hélice de ADN. Ahora, esta hélice de ADN se inserta en cualquier sitio del cromosoma del huésped. Finalmente, se transcribe este ADN en la célula huésped produciendo muchas copias del ARN original del virus que se traducirán para formar las proteínas de la cápside y de la envoltura.

2.2.4.2. Plásmidos

Las bacterias, a parte del cromosoma, contienen moléculas de ADN ajenas al cromosoma llamadas plásmidos [30][1]. Un plásmido es un fragmento de ADN circular de doble hélice que se replica y se transcribe de forma independiente al cromosoma. Este fragmento de material genético puede trasmitirse de una célula a otra permaneciendo inmutable el cromosoma. En cuanto al tamaño, los plásmidos tienen tamaño variable desde menos de 10 hasta 400 pares de kilobases. Además, una bacteria no tiene por

qué tener un solo plásmido, de hecho puede tener varias copias del mismo plásmido o distintos plásmidos, aunque es difícil encontrar plásmidos con cierta semejanza de ADN entre sí dentro de la misma bacteria que no hayan sido creados a partir de una copia.

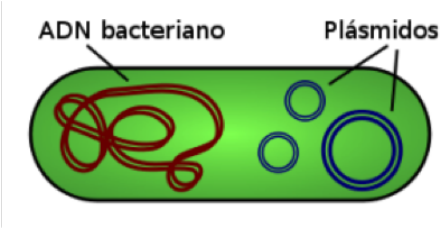


Figura 2.7: Plásmido [36]

Los plásmidos, además de codificar resistencia a los antibióticos, pueden codificar factores de virulencia, lo que puede causar que la bacteria resulte patógena. El gran interés que muestra la biología sintética en los plásmidos proviene de la idea de que los plásmidos pueden ejercer la función de cables intercelulares para crear un sofisticado circuito multicelular. Los plásmidos pueden ser programados, por ejemplo, con puertas lógicas y así conseguir que hagan la función de antibióticos programables o de plásmidos centinela que detecten plásmidos peligrosos en bacterias virulentas. El proceso biológico que utilizan los plásmidos para ejercer esa función de cable intercelular es la conjugación bacteriana. El proyecto europeo PLASWIRES se basa en estas herramientas para poder crear una colonia de bacterias que realicen tareas, se comuniquen y resuelvan problemas de manera paralela y distribuida.

Conjugación

La conjugación bacteriana es el proceso de transferencia horizontal de genes desde una célula donante a otra receptora [30][1]. Normalmente, en la transferencia únicamente se transmite el plásmido, pero hay casos en que estos plásmidos se comportan como episomas. En este caso, se trasmite parte del cromosoma además del plásmido. Esta parte del cromosoma se adherirá en el cromosoma de la receptora y se formará en ella una combinación con el cromosoma nuevo.

Este proceso requiere un contacto directo entre las dos bacterias que intervienen en el proceso. Toda bacteria E. Coli esta formada por una serie de filamentos cortos y

estrechos, que son huecos y rectos llamados pilus. Estos pilus están compuestos por proteínas y su función principal es mediar la adhesión. Existen dos tipos:

- Sexuales: (1-10 por célula). Tienen como función específica la transmisión de material genético en el proceso de conjugación.
- No sexuales: (100-200 por célula). Tienen como función específica la adhesión a tejidos y superficies.

Una vez que el pilus se ha adherido a otra bacteria se produce el proceso de conjugación. Durante este proceso, una de las cadenas de ADN del plásmido de la bacteria donante se desenrolla y se transmite a través del pilus a la bacteria receptora. En el interior de la bacteria receptora se arma la primera hebra circular. Al mismo tiempo se sintetiza la segunda hebra circular regenerándose en ambas bacterias el plásmido completo.

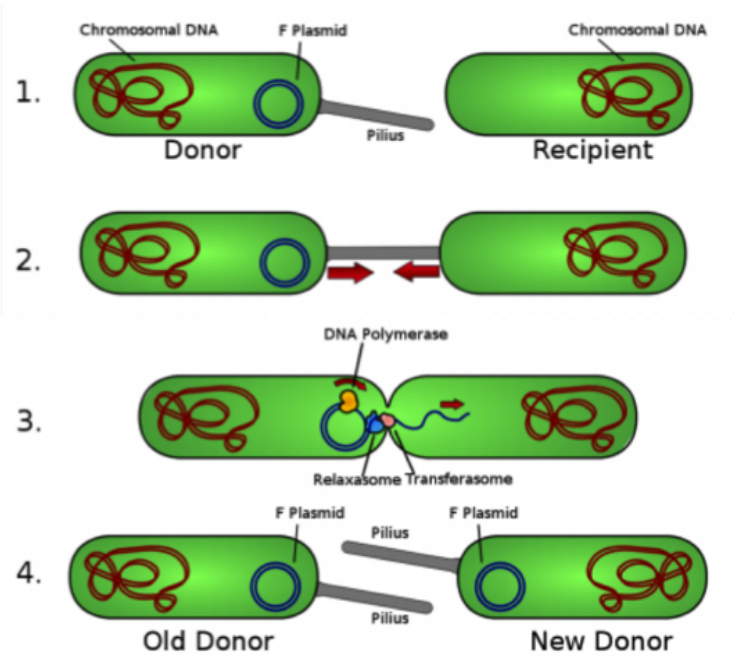


Figura 2.8: Proceso de conjugación [36]

En este momento, la bacteria receptora pasa a ser transconjugada teniendo también posteriormente la capacidad de conjugar, comportándose como donante.

2.3. Expresión y regulación genética

Los genes, a través de la transcripción y la traducción, dirigen la síntesis de proteínas, muchas de las cuales funcionan como enzimas, utilizadas para el metabolismo celular. Como este proceso requiere un enorme gasto de energía para la célula, su regulación es importante. La célula conserva energía mediante la síntesis exclusiva de las proteínas necesarias en un momento particular.

Muchos genes, tal vez del 60 al 80 %, no están regulados sino que son constitutivos, lo que significa que sus productos se producen de manera constante con una tasa fija. Por lo general, estos genes codifican las enzimas que la célula necesita en cantidades bastante grandes para sus principales procesos vitales. La producción de otras enzimas es regulada para que estén presentes solo cuando se las necesita.

2.3.1. Mecanismos de Control

Existen dos mecanismos de control genético conocidos como represión e inducción. Estos mecanismos regulan la transcripción de mRNA y por consiguiente la síntesis de enzimas. Concretamente, controlan la formación y la cantidad de enzimas en la célula, no las actividades de las enzimas.

2.3.1.1. Represión

El mecanismo regulador que inhibe la expresión genética y disminuye la síntesis de enzimas se denomina represión. La represión provoca una disminución de la velocidad de síntesis de las enzimas que conducen a la formación de ese producto. La represión está mediada por proteínas reguladoras denominadas represores, que bloquean la capacidad de la ARN polimerasa de iniciar la transcripción de los genes reprimidos. En ausencia de este represor, el gen reprimible está activado.

2.3.1.2. Inducción

El proceso que activa la transcripción de un gen o de varios genes es la inducción. Una sustancia que induce la transcripción de un gen se denomina inductor y las enzimas que se sintetizan en presencia de los inductores son enzimas inductibles. En ausencia de inductor, un gen inductible está inactivo.

2.3.2. Riboswitchs

Los riboswitchs regulan la expresión bloqueando la traducción de ARN. La adición de una horquilla a la zona de unión del ribosoma al mRNA previene que el ribosoma se una eficientemente. Esta horquilla puede ser evitada en presencia de un pequeño ARN regulador. En este caso se expone la zona de unión del ribosoma y se activa la expresión (fig. 2.9).

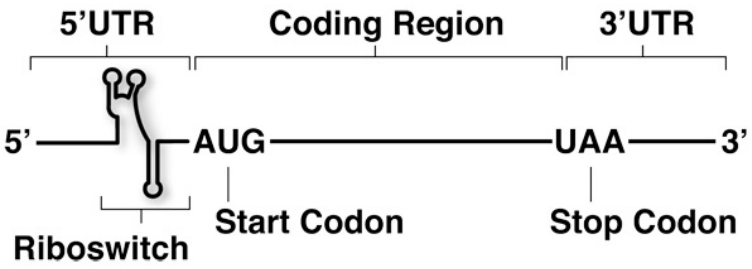


Figura 2.9: Riboswitch [15]

2.3.3. Moléculas inductoras/inhibidoras

La actividad de la enzimas puede ser afectada por otras moléculas. Estas moléculas se unen a las enzimas y pueden producir dos respuestas diferentes, incrementar su actividad (inducirlas) o disminuir su actividad (inhibirlas).

La unión de una molécula inductora a una enzima produce un incremento de la velocidad de las reacciones canalizadas por las enzimas. Algunas enzimas requieren la presencia de estas moléculas para realizar alguna actividad. Por tanto, un factor de

transcripción que necesite la presencia de una molécula inductora para adherirse al promotor, no podrá unirse hasta que esta molécula se encuentre en el medio.

Por otro lado, la unión de una molécula inhibidora a una enzima puede impedir la entrada del sustrato al sitio activo de la enzima o evitar que la enzima catalice su reacción correspondiente. De modo que en presencia de esta molécula inhibidora, la proteína que actuar como factor de transcripción, no puede adherirse a un promotor y provoca que el gen que regula ese promotor no se active.

Capítulo 3

Circuitos Genéticos

3.1 Introducción 23
3.2 Partes de los circuitos genéticos 25
3.2.1 Sensores 26
3.2.2 Circuitos 26
3.2.3 Actuadores 27
3.3 Circuitos básicos 28
3.3.1 Toggle Switch 28
3.3.2 Repressilator 29

3.1. Introducción

La ingeniería genética con ADN recombinante es una tecnología potente que permite a los biólogos rediseñar formas de vida mediante la modificación o extensión de su ADN. Algunos avances en este campo han permitido la creación de biomateriales, productos farmacéuticos sintéticos o detección de toxinas mediante centinelas sintéticos.

Los ingenieros genéticos usan genes y proteínas como bloques que se pueden combinar entre ellos para crear nuevos tipos de células o nuevas funcionalidades. La biológica sintética, el campo dedicado a averiguar como combinar genes de forma interesante y novedosa, requiere un amplio conocimiento biológico, habilidades creativas en ingeniería

y expertos informáticos. Mediante esta unión de expertos con diferentes habilidades, se pueden crear nuevas formas de vida capaces de realizar tareas concretas.

El código genético es como cualquier otro lenguaje, para escribirlo hace falta aprenderlo y entenderlo. En los últimos siglos, los científicos han sido capaces de aprender a leer el código genético que todas las células contienen. Se ha investigado que genes determinan qué características de las células y organismos, y como modificaciones en estos genes pueden alterar esas características.

El descubrimiento de lógica matemática en regulación genética en 1960 (eg. lac operon; Monod and Jacob, 1961)[23] y los nuevos logros conseguidos en ingeniería genética desde 1970, como el ADN recombinante, han pavimentado el camino hacia la Biología Sintética. La Biología Sintética extiende el espíritu de la ingeniería genética centrándose en sistemas completos de genes y sus producciones. Centrarse en sistemas de genes en lugar de genes individualizados se comparte con otra disciplina denominada Biología de Sistemas, que analiza organismos biológicos e intenta construir modelos matemáticos que se adapten a su comportamiento.

El objetivo de la Biología Sintética consiste en extender y modificar el comportamiento de organismos para que ejecuten nuevas tareas. Una forma de ver los objetivos y métodos de la biología sintética es contextualizarlo en el ámbito de la informática (fig. 3.1). El diseño de un nuevo comportamiento ocurre en la parte superior de la jerarquía pero el proceso debe comenzar desde la parte inferior. Al principio de la jerarquía se encuentran el ADN, ARN, proteínas y metabolitos (incluyendo lípidos, carbohidratos, amino ácidos y nucleótidos), análogamente a los transistores, resistencias y condensadores de informática. La siguiente capa esta formada por las reacciones químicas que regulan el flujo de información y manipulan los procesos físicos, de forma equivalente a las puertas lógicas que se ejecutan en un cómputo. En la capa de módulos, se utiliza un librería con una gran variedad de elementos biológicos que ensamblándolos se consiguen crear circuitos mas complejos. Estos circuitos se insertarán dentro de la célula para usar su propia maquinaria para ejecutar esos circuitos.

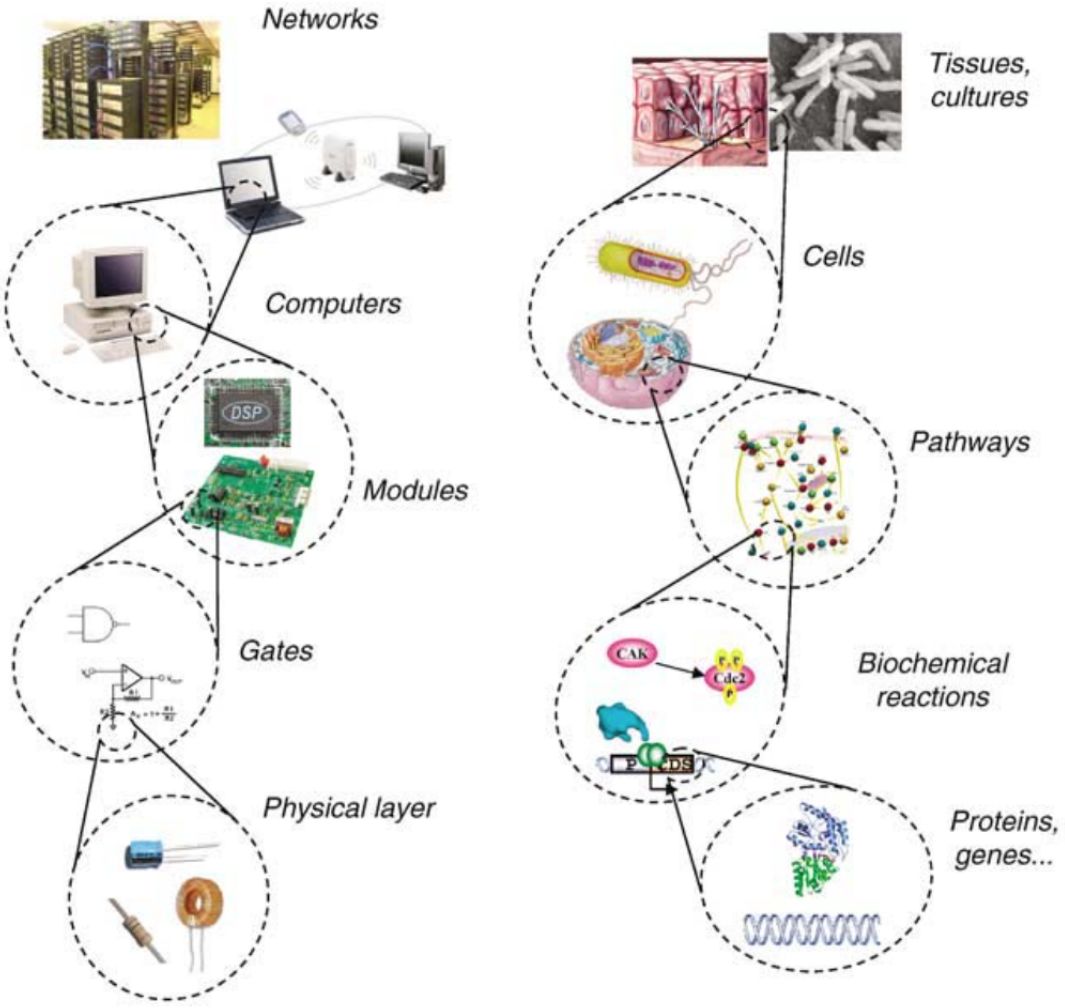


Figura 3.1: Comparación Biología-Informática [3]

3.2. Partes de los circuitos genéticos

Las partes utilizadas en el diseño de circuitos genéticos se han dividido en tres categorías principales [42]. Los sensores abarcan los medios por los cuales la célula recibe información. Los circuitos genéticos representan cómo se procesa la información recibida y las decisiones a tomar. Los actuadores describen cómo se usan los circuitos y sensores para controlar los procesos en la célula.

3.2.1. Sensores

Las células utilizan los sensores para poder identificar un microambiente, para dirigir la comunicación entre células o para que el sistema responda a comandos externos. Los sensores pueden ser conectados entre ellos para activar o desactivar genes o para influir directamente en el comportamiento celular.

Las células cambian sus patrones de expresión genética en respuesta a diferentes condiciones ambientales. Hay varias condiciones, tales como pH, temperatura, concentración de oxígeno o la luz UV, para los cuales las bacterias tienen sensores. Los promotores que se activen bajo estas condiciones se pueden usar como entradas sensoriales.

3.2.2. Circuitos

Los circuitos genéticos permiten a las células procesar las señales de entrada, tomar decisiones lógicas, disponer de memoria y comunicarse unas con otras. Una variedad de circuitos sintéticos se han construido para imitar la capacidad de procesamiento de información de los circuitos electrónicos, tales como inversores, puertas lógicas, generadores de impulsos y osciladores. Estos circuitos se pueden utilizar para convertir una salida del sensor en una respuesta biológica, además de programar a la célula para realizar una serie de tareas coordinadas.

Los interruptores genéticos (switches) se utilizan para activar la expresión de genes una vez que una entrada ha cruzado el umbral requerido para la activación. Un interruptor también puede ser utilizado como intermediario entre la salida de un sensor y una respuesta biológica. Este mecanismo puede construirse utilizando inductores. Un inversor es un interruptor que produce una respuesta recíproca. Cuando la entrada al inversor está encendida, la salida está apagada, y viceversa.

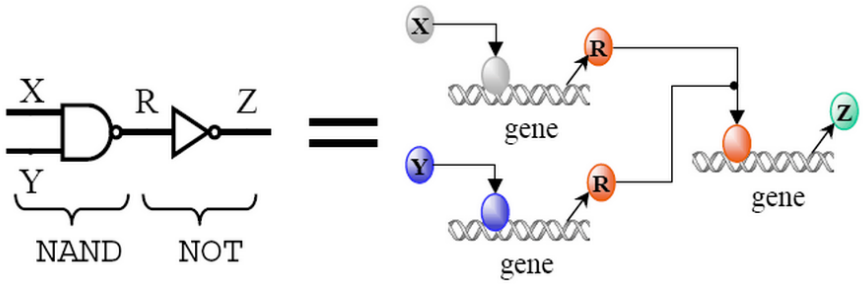


Figura 3.2: Diseño de una puerta NAND conectada a un inversor [44]

Las puertas lógicas son las herramientas más usadas en la construcción de circuitos digitales. Aplican una operación computacional para convertir una o varias entradas en una única salida. Por ejemplo, la salida de una puerta AND solo ocurre cuando las dos entradas están presentes. Por el contrario, la salida de una puerta OR ocurre cuando una de ellas (o ambas) están encendidas. Se han probado diversas puertas lógicas en circuitos genéticos, incluyendo las puertas NOT, AND, OR y NOR.

3.2.3. Actuadores

El comportamiento celular puede ser controlado usando las salidas de circuitos genéticos sintéticos para ejecutar respuestas naturales o transgénicas. Estas respuestas pueden ser funciones específicas a realizar por la célula o un cambio en su metabolismo o apariencia.

Uno de los actuadores más usados es el reporter GFP. Esta proteína produce un brillo verde fluorescente en la bacteria permitiendo comprobar visualmente un estado concreto del circuito.

Entre las funciones que pueden realizar las células a causa de un circuito genético, están las funciones de conjugar, morir o modificar su tasa de crecimiento entre otras.

3.3. Circuitos básicos

3.3.1. Toggle Switch

Un *Toggle Switch* [17] es una red sintética de genes que se mantiene en estados biestables. Mediante la inducción de señales químicas o térmicas puede cambiarse de un estado a otro. El circuito genético comprende dos represores, R1 y R2, sus promotores P1 y P2 y sus inductores I1 y I2 que desactivan los represores (fig. 3.3). La representación del circuito consiste en dos puertas de implicación interconectadas.

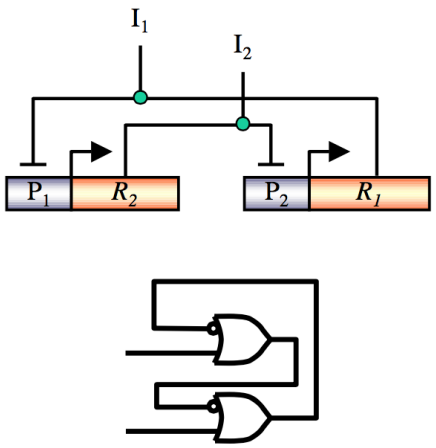


Figura 3.3: Circuito Toggle Switch [17]

El comportamiento del circuito consiste en que cuando el primer gen está activo, produce una proteína represora que inhibe la expresión del segundo gen. Y cuando el segundo gen está activo, produce una proteína represora que inhibe la expresión del primer gen. Como resultado, ambos genes no pueden estar activos al mismo tiempo. Si un científico agrega un inductor que desactiva el primer gen, esto permite al segundo gen activarse, manteniendo el primer gen desactivado. Si se agrega un inductor que que desactiva el segundo gen, esto permite al segundo gen activarse, manteniendo el segundo gen desactivado.

El toggle switch es un circuito sumamente interesante ya que da a las células una

memoria. Antes del toggle switch, si los científicos deseaban activar y desactivar un gen de una célula, debían agregar continuamente un inductor que active un gen que codifique esa proteína. Ocurriría lo mismo si al tener que mantener una luz encendida tuviéramos que mantener el dedo en el interruptor continuamente. El toggle switch, sin embargo, mantiene un gen activado agregando un inductor una única vez. Este circuito da a la célula una memoria del estado en el que debe mantenerse.

3.3.2. Repressilator

El *Repressilator* [16] es una red genética oscilante construida con tres represores. La frecuencia de oscilación de la red genética es menor que la frecuencia de división de las células, y como resultado, las oscilaciones se propagan a través de las generaciones. La red comprende tres tipos de represores CI, LacI y TetR, y sus correspondientes promotores. CI reprime la expresión de LacI, el cual a su vez reprime la expresión de TetR. La proteína TetR, de la misma forma, inhibe la expresión de CI (fig. 3.4). Una vez se han definido correctamente las características entre todos los genes y se han definido tiempos de apropiados de expresión, este sistema oscilará con una periocidad y amplitud regular.

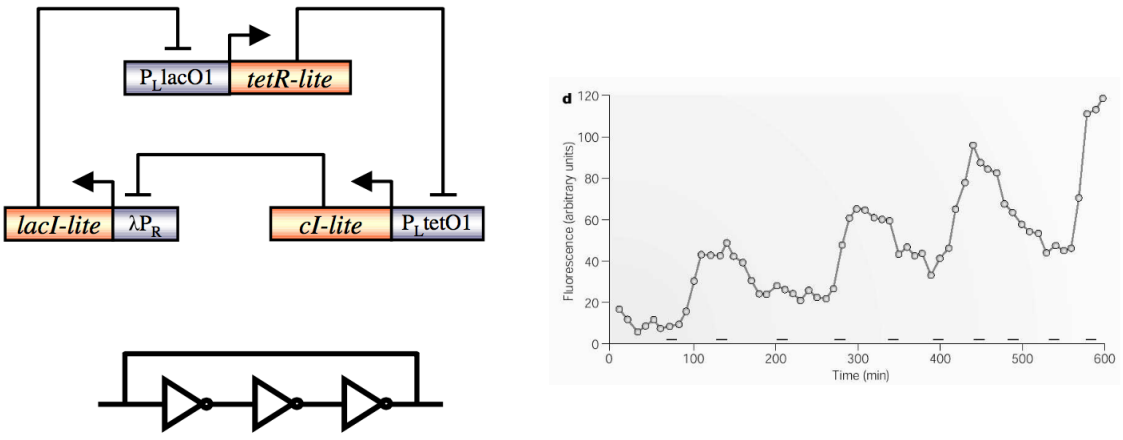


Figura 3.4: Circuito Repressilator [16]

Con el fin de sensar esta oscilación, se añade otra represión al circuito. La proteína tetR reprime la proteína GFP. La proteína GFP emite una luz verde fluorescente que permite visualizar el estado del oscilador. Dado que tetR reprime las proteínas CI y GFP, esta última reportará el estado aproximado de CI. El gráfico de la figura 3.4 muestra la oscilación de la proteína GFP medida en una sola célula en el tiempo.

Experimentalmente se ha observado que, aunque todas las bacterias de la colonia contienen el mismo circuito genético, no todas presentan el mismo periodo y amplitud en las oscilaciones. Esta variabilidad es atribuida a la naturaleza estocástica de las reacciones bioquímicas que controlan la maquinaria de la expresión genética. Varios proyectos de investigación están intentando actualmente implementar redes genéticas sintéticas que exhiban una oscilación mejorada en la que los periodos y amplitudes de las oscilaciones en todas las bacterias sea regular.

Capítulo 4

Estado del Arte

4.1 Simuladores de colonias bacterianas 31
4.1.1 CellModeller 32
4.1.2 iDynomics 33
4.1.3 BactoSIM 33
4.1.4 BacSIM 33
4.1.5 BSIM 34
4.1.6 DiSCUS 34
4.1.7 GRO 34
4.2 Herramientas para el diseño de circuitos genéticos 37
4.2.1 GenoCAD 37
4.2.2 EuGene 38
4.2.3 GEC 38
4.3 Modelos de Sistemas de Regulación Genética 39
4.3.1 Redes Booleanas 39
4.3.2 Ecuaciones Diferenciales y en diferencias 40

4.1. Simuladores de colonias bacterianas

Los simuladores de colonias de bacterias se basan en dos modelos diferentes, el modelo AbM/IbM (Agent/Individual based Model), que trata cada entidad de forma

individualizada, y los modelos que analizan las dinámicas a nivel de población con ODEs/PDEs (Ordinary Differential Equations/ Partial Differential Equations). Representar comportamientos en colonias bacterianas mediante ODEs puede resultar más preciso biológicamente pero también computacionalmente costoso debido a la complejidad de los cálculos a realizar. Por otro lado los modelos basados en individuos no requieren tanta carga computacional, excepto en colonias con una gran cantidad de individuos para los que existen soluciones mediante computación paralela, aunque no representen exactamente los resultados experimentales. En este proyecto nos centraremos en los sistemas multiagente en el que cada bacteria actúa de forma individualizada e interactúa con las demás. Existen numerosos simuladores de colonias bacterianas basados en esta metodología, pero en el ámbito de la Biología Sintética son siete los más utilizados.

4.1.1. CellModeller

CellModeller [13] es un módulo de Python para el análisis multicelular 2D diseñado por la Universidad de Cambridge. Este módulo no fue creado especialmente para simular el comportamiento de bacterias, su función es la de simular el comportamiento de células en general, que abarca desde algas microscópicas hasta las células que intervienen en la formación de tejidos. En cambio, ofrece una gran variedad de herramientas para facilitar la configuración del usuario.

Recientemente, se le ha agregado un nuevo módulo de OpenCl para permitir la computación paralela. La computación paralela permite conseguir una colonia con más número de bacterias en menos tiempo de simulación. Este software incluye modelos de la biofísica, la genética y la señalización intercelular. A partir de una sola célula consigue simular una colonia de aproximadamente 32.000 células individuales en 30 min. Reproduce las características principales de una colonia como el crecimiento exponencial del número de células permitiendo añadir limitaciones físicas duras o blandas. Posteriormente, se implementó CellModeller4 [34], que permite la especificación del comportamiento de la célula a través de modelos basados en reglas y ecuaciones diferenciales.

4.1.2. iDynomics

iDynoMICS (individual-based Dynamics of Microbial Communities Simulator) [26] es un simulador de código abierto que permite trabajar con varios modelos computacionales de bacterias, en 2D y 3D. Permite configurar una gran cantidad de parámetros respecto al entorno y reacciones químicas. En 2011 se agregó un primer módulo con la capacidad de trabajar con bacterias en forma capsular, ya que inicialmente solo permitía el uso de bacterias en forma esférica [6]. Esto causó la necesidad de diseñar un nuevo algoritmo para los solapamientos de las bacterias. Posteriormente, se le añadió un segundo módulo que permitió la simulación de puertas AND mediante conjugación [7]. Actualmente está en fase de verificación, puesto que no se ha comprobado que funciona de una forma biológicamente correcta.

4.1.3. BactoSIM

BactoSim [5] es un simulador espacial implementado en Java. Fue creado específicamente para simular el comportamiento de bacterias en una colonia. En este simulador, se define el espacio como una rejilla discreta en la que se colocan todos los agentes y interaccionan entre ellos. Además de tener todas las funcionalidades en cuanto a crecimiento e interacciones, posee la funcionalidad de conjugación con el propósito de estudiar este proceso *in-silico*. Este simulador está siendo continuamente mejorado para validarse con los datos experimentales del IBBTEC.

4.1.4. BacSIM

BacSim [24] es un simulador multiagente en el que cada agente es en realidad una esfera que se representa en un espacio 2D como un círculo. Este simulador no admite la forma de bacilo en sus células, lo que afecta en gran medida al comportamiento de la colonia. El modelo describe propiedades bacterianas incluyendo captación de sustrato, el metabolismo, el mantenimiento, la división celular y la muerte en el nivel de célula individual. Para la difusión de sustrato, se utiliza una red de difusión de dos dimensiones. Con el fin de hacer el modelo fácilmente aplicable a diferentes tipos de bacterias, este software cuenta solo con ocho parámetros que definir y que pueden ser aleatoriamente definidos.

4.1.5. BSIM

BSim [19] es una herramienta de modelado altamente personalizable basada en agentes. Se basa en un espacio 3D en el que las bacterias son definidas de forma esférica. Fue creado con la idea de que todos los aspectos del modelo deben ser actualizables, ampliados o reemplazados por versiones definidas por el usuario. Este enfoque permite el rápido desarrollo de simulaciones que capturan las características principales de un sistema, al tiempo que permite que estos sean redefinidos por datos experimentales según estén disponibles.

4.1.6. DiSCUS

DiSCUS (Discrete Simulation of Conjugation Using Springs) [4] fue, junto con BactoSim, de los primeros en ofrecer en sus simulaciones redes genéticas y comunicación intercelular mediante conjugación. Las células se definen en forma de bacilo y vienen dadas por una longitud y un radio. El proceso de conjugación se representa mediante un muelle elástico que conecta a una donante con una receptora, este muelle está constantemente monitorizado durante un tiempo establecido de conjugación, en el que se comprueba que está constantemente conectado con ambas células. Además, se puede establecer si las células transconjugadas tienen o no la capacidad de donar plásmidos.

4.1.7. GRO

GRO [37] es un paquete de software de código abierto que combina un enfoque de sistemas distribuidos y computación paralela con la simulación de miles de bacterias que crecen en un entorno 2D. Fue creado específicamente para el ámbito de la Biología Sintética. Este simulador reproduce el movimiento, el crecimiento y las divisiones de las células. Además, permite el uso de señales que afectan al comportamiento de ciertas sustancias químicas sobre las bacterias.

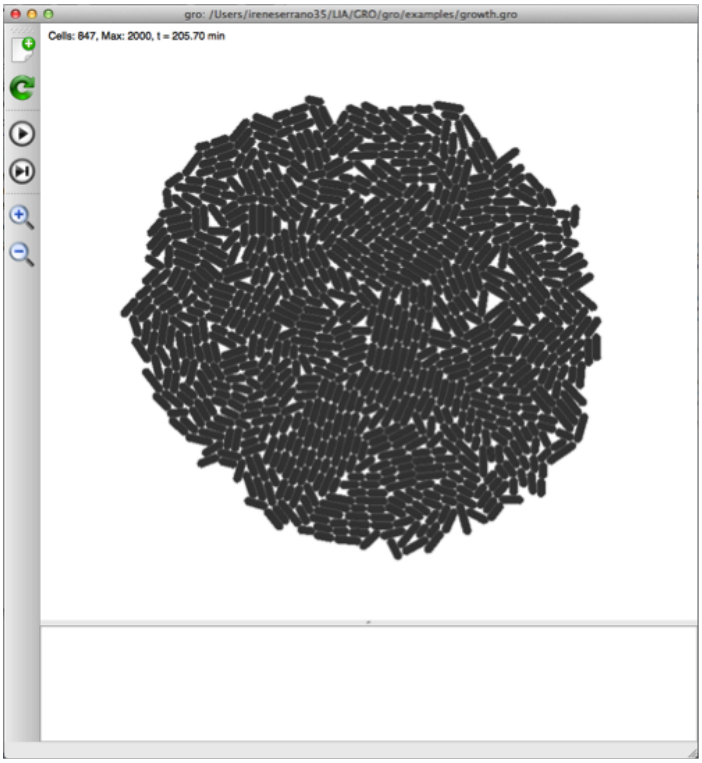


Figura 4.1: GRO

Recientemente, se han implementado en GRO las dinámicas de comunicación intercelular mediante conjugación [36]. Permitiendo el traspaso de plásmidos de una bacteria a otra.

4.1.7.1. Parametrización

Las células se definen inicialmente con un diámetro de 1 μm y un largo de 2 μm lo que resulta un volumen de 1.57 fL. Normalmente, las bacterias crecen 0.034 fL/min, por eso ese es el valor por defecto que utiliza GRO para el crecimiento, pero este se puede definir al principio de la simulación.

GRO resume la variación de volumen en la fórmula $V^{'} = V + \delta kV$ donde δ es aproximadamente la variación de tiempo, que de aquí en adelante denominaremos dt. Este dt es también la duración de una iteración en la simulación, por eso, si se define una variación de tiempo muy alta se consiguen simulaciones menos precisas, pero si en cambio

se define un dt muy bajo, el crecimiento es más lento pero se actualizan los estados de las células en intervalos más cortos consiguiendo una mayor precisión.

Cada bacteria crece hasta que aproximadamente dobla su tamaño, en este punto se divide. El volumen en el que las bacterias se dividen por defecto es de 3.14 fL con una varianza de 0.005 fL^2. Ambos parámetros pueden ser definidos antes de comenzar una simulación.

Para el caso de la conjugación, se puede definir la cantidad de conjugaciones que se van a dar en cada célula en cada ciclo de vida y el plásmido que se va a conjugar, en caso de que estén presentes varios plásmidos en una simulación.

4.1.7.2. Lenguaje GRO

Este simulador tiene su propio lenguaje de programación denominado igual que él, *gro*. Se trata de un lenguaje funcional bastante sencillo basado en un conjunto de reglas que posteriormente el interpreta. Este lenguaje permite definir todos los parámetros que serán dados a la colonia para su posterior simulación. A continuación se muestra un ejemplo del lenguaje en el que se define la variación de tiempo dt a 0.1 y la posición de una nueva bacteria E. Coli, que en este caso la posiciona en el punto (0,0). Una vez ejecutado este ejemplo, solo hay que esperar unos minutos para que GRO consiga crear una colonia de miles de bacterias.

```
include gro
set ( "dt", 0.1 );
program p() := {
      skip();
};
ecoli ( [ x := 0, y := 0 ], program p() );
```

4.2. Herramientas para el diseño de circuitos genéticos

Las cadenas de ADN están formadas por segmentos como promotores, factores de transcripción, genes, proteínas y otros. En la mayoría de las herramientas dedicadas al diseño de circuitos genéticos a estos segmentos se les conoce como "partes". Las partes necesitan estar en un orden determinado y próximas unas a otras para asegurar una correcta transcripción y traducción genética. La combinación de estas partes hace que se puedan diseñar circuitos genéticos más complejos. El único inconveniente es que aún hoy en día no se conocen completamente todos los mecanismos que intervienen en la expresión genética. Una forma de formalizar este proceso es diseñando herramientas que guíen al usuario en el diseño de estos circuitos para verificar los datos recogidos por el momento sobre este proceso.

4.2.1. GenoCAD

GenoCAD [10][12][45] (www.genocad.org) es una aplicación web gratuita para diseñar vectores de expresión de proteínas, redes artificiales de genes y otros circuitos genéticos formados por distintas partes. El diseño de circuitos en GenoCAD está basado en reglas que definen las combinaciones entre las distintas partes. Estas reglas se definen gracias a las gramáticas de GenoCAD, que aseguran que cada regla contenga alguna parte cada una de las cuatro categorías básicas: Promotores, RBS (Ribosome Binding Site), Genes y Terminadores.

Las partes tienen un identificador único, un nombre, una descripción más general y una secuencia de ADN. Las piezas se asocian con una gramática y se asignan a una categoría piezas tales un promotor, gen, etc. GenoCAD.org tiene una librería con miles de partes básicas distintas denominada BioBrickTM, donde están almacenados los parámetros que definen estas partes.

4.2.2. EuGene

EuGene [8][9] (www.eugenecad.org) es un software para el rediseño de genes sintéticos. Permite analizar genes y proteínas para rediseñados de acuerdo a diversos factores biológicos como el uso y el contexto del codón, repeticiones de nucleótidos o secuencias perjudiciales.Además, facilita las tareas comunes relacionadas con la caracterización de genes y su manipulación, tales como la identificación de genes y genoma o predicción de la estructura secundaria de la proteína.

Numerosas bases de datos biológicas proporcionan los medios para buscar y recuperar sus datos. EuGene se aprovecha de estos recursos para obtener la máxima calidad de información acerca de un gen. Además, otras herramientas se utilizan para realizar cálculos complejos, tales como la realización de las alineaciones y la predicción de las estructuras secundarias de las proteínas. Esto se realiza de manera automática, sin problemas para el usuario, y se procesan los resultados antes de ser mostrados. Por otra parte, EuGene usa algoritmos genéticos para mejorar la optimización de genes y así permitir los mejores resultados.

4.2.3. GEC

El objetivo principal de GEC [41][32] (http://research.microsoft.com/gec) es facilitar el diseño, análisis y aplicación de dispositivos biológicos dentro de células vivas. GEC se basa en investigaciones previas en el campo de la Biología Sintética, incluyendo un registro de partes estándar (http://partsregistry.org) junto con técnicas experimentales para combinar piezas. La principal innovación detrás de GEC es tomar el diseño desde un paso más allá, permitiendo crear dispositivos biológicos con poco o ningún conocimiento de las partes específicas disponibles. El usuario sólo necesita tener conocimientos de los tipos de partes disponibles, promotores, ribosomas, regiones de codificación de proteínas y terminadores. Una vez el dispositivo biológico ha sido diseñado, GEC determina automáticamente el conjunto de piezas reales que satisfacen las restricciones del diseño. GEC compila cada una de las soluciones a un conjunto de reacciones químicas que pueden ser analizadas por el usuario. Las soluciones que presenten el comportamiento deseado podrán ser sintetizadas en células reales. Aunque no hay garantía de que los resultados que produce esta herramienta sean los mismos que produce una célula

real.

4.3. Modelos de Sistemas de Regulación Genética

Los métodos usados para la modernización de los sistemas de regulación genética incluyen el método booleano, que describe el estado de los genes simplemente con "ON" y "OFF", y el método continuo que usa ecuaciones diferenciales ordinarias (EDOs) para describir las concentraciones de los productos de los genes en el tiempo [35]. El método booleano es favorable para la formulación y computación, mientras el método continuo tiene la ventaja de ser más preciso físicamente.

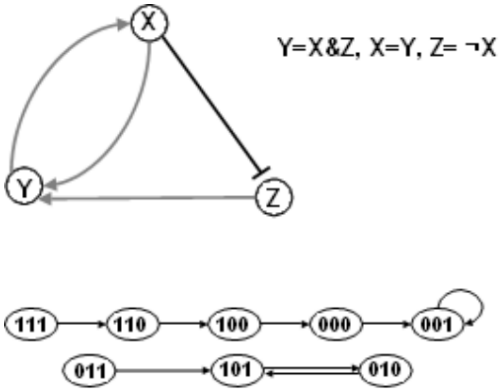
4.3.1. Redes Booleanas

El modelo dinámico más simple (redes booleanas síncronas), fue usado para modelar la regulación de genes en los años 60 por Stuart Kauffman [25].

Las redes booleanas, parten de la idea de que interruptores binarios on/off funcionando en una sucesión discreta de instantes de tiempo, pueden describir importantes aspectos de regulación genética. En las redes booleanas síncronas, todos los genes cambian su estado de manera simultánea.

El estado de la red viene dado por una n-tupla de ceros y unos, que indican los genes que en ese instante se expresan (encendidos) o no (apagados). Conforme avanza el tiempo, el estado de la red va cambiando de un estado a otro. Para una red de n genes, existe un total de 2^n posibles estados diferentes, por ejemplo, para una red de tres genes, los estados posibles son (0,0,0), (0,0,1),...,(1,1,1). Sin embargo, existen estados a los que nunca se llega. También existen atractores, estados o conjuntos de estados, en los que una vez alcanzados no cambian.

Por ejemplo, en la figura 4.2 existen dos atractores, uno simple es el estado (0,0,1), y el otro compuesto por la alternancia de los estados (1,0,1) y (0,1,0).



t	t+1
X Y Z	X Y Z
0 0 0	0 0 1
0 0 1	0 0 1
0 1 0	1 0 1
0 1 1	1 0 1
1 0 0	0 0 0
1 0 1	0 1 0
1 1 0	1 0 0
1 1 1	1 1 0

Figura 4.2: Ejemplo de red booleana síncrona [35]

4.3.2. Ecuaciones Diferenciales y en diferencias

Las ecuaciones diferenciales y en diferencia, permiten una detallada descripción de este aspecto, modelando explícitamente los cambios de concentraciones moleculares a través del tiempo [11][14][22][28][43].

El modelo básico de ecuación en diferencia es de la forma:

$$g_1(t+\Delta t) - g_1(t) = (w_{11}g_1(t) + ... + w_{1n}g_n(t))\Delta t$$
$$...$$
$$g_n(t+\Delta t) - g_n(t) = (w_{n1}g_1(t) + ... + w_{nn}g_n(t))\Delta t$$

donde $g_i(t+\Delta t)$ es el novel de expresión del gen i en el instante $t+\Delta t$, Y w_{ij} un peso indicando la influencia del gen j sobre la expresión del gen i, con *i,j=1...n.*

Este modelo asume una lógica de control lineal, ya que el nivel de expresión de un gen en el instante $t + \Delta t$ depende linealmente de los niveles de expresión de todos los genes en el instante t. No obstante, para cada gen se pueden añadir términos adicionales que indiquen la influencia de otras sustancias [14].

Las ecuaciones diferenciales son similares a las ecuaciones en diferencia, solo que el cambio de concentración se produce de manera continua, y contemplando la diferencia temporal entre dos instantes consecutivos como un incremento infinitesimal (Δt tiende a 0).

Una cuestión importante de los modelos basados en ecuaciones diferenciales y en diferencia, dependen de parámetros numéricos que a menudo son difíciles de obtener de manera experimental.

Capítulo 5

Solución Propuesta

5.1 Modelo booleano probabilístico de regulación genética 44
5.1.1 Operones 45
5.1.2 Puertas Lógicas 45
5.1.3 Activaciones/degradaciones de proteínas y ARNs 48
5.1.4 Riboswitchs y moléculas inductoras 49
5.1.5 Estado inicial 50
5.1.6 Ruido 50
5.1.7 Actuadores 51
5.2 Diseño de circuitos genéticos 51
5.2.1 Parámetros 52
5.2.2 Matriz de ARNs-riboswitchs 52
5.2.3 Matriz de moléculas inductoras 53
5.2.4 Operones 54
5.2.5 Tiempos de degradación 57
5.2.6 Matriz de plásmidos 58
5.2.7 Acciones 59
5.2.8 E. Coli 60
5.2.9 Agregar molécula inductora 60
5.3 Procedimiento 61

Dado que GRO es un simulador basado en el modelo IbM es necesario simular la expresión genética desde el nivel del individuo. En colonias con un gran número de individuos, las EDOs supondrían una gran carga computacional teniendo que procesar todos los cálculos al menos una vez por célula. Por esta razón, se ha implementado una red booleana asíncrona en cada bacteria, a la que se le han añadido probabilidades de acierto y ruido en sus procesos.

Los investigadores de biología sintética necesitan poder visualizar el comportamiento de una colonia en presencia de un circuito biológico sintético. El diseño de estos circuitos debe ser una característica indispensable en los simuladores dedicados a este campo. Para ello se ha ampliado el lenguaje *gro* de manera que sea posible diseñar plásmidos con circuitos genéticos. Estos plásmidos se dispondrán en células concretas y, mediante el módulo de expresión genética, se visualiza lo ocurrido en su presencia. Gracias a la capacidad del simulador de manejar las dinámicas de conjugación, estos plásmidos se traspasarán de una célula a otra y, dependiendo de las condiciones, los resultados en cada una de ellas podrán ser diferentes debido a que cada célula tiene su propia red booleana.

5.1. Modelo booleano probabilístico de regulación genética

Se ha diseñado un modelo de expresión genética basado en concentraciones binarias de proteínas. El estado de las proteínas se representa como *on*(1) si está presente o *off*(0) en su ausencia. Cada bacteria tiene una lista con el estado de sus propias proteínas internas. De esta forma, si en un experimento van a estar presentes, por ejemplo, 5 proteínas, un ejemplo de una lista de proteínas en una célula es (1,0,1,0,0), indicando que tiene presentes la *proteína1* y la *proteína3*. Cada Proteína tiene asociado su propio ARN. Por lo tanto, de la misma forma que con la lista de proteínas, cada célula tiene una lista booleana de ARNs con las cadenas de ARN que se han transcrito para sintetizar posteriormente su proteína correspondiente. La proteína representada en la primera posición de la lista de proteínas tiene asociado al ARN situado en la primera posición de la lista de ARNs. Ya que esa cadena de ARN es la que posteriormente se

traducirá en la proteína situada en la misma posición de la lista.

En los circuitos genéticos están disponibles las puertas lógicas *yes, and* y *or*. Estas puertas tienen como entrada ciertas proteínas elegidas como factores de transcripción. La presencia de estos factores hace que el promotor del circuito se active generando como salida la activación o represión de una o varias proteínas.

5.1.1. Operones

Un operón está formado por el conjunto de promotor, genes y terminador. Uno o varios operones conforman los circuitos genéticos de los que están formados los plásmidos. El promotor se activa mediante uno o varios factores de transcripción. Las puertas lógicas *yes, and* y *or* condicionan la activación del promotor. Los operones pueden estar formados por uno o varios genes. La salida de un operón activa o reprime una o varias proteínas concretas.

Un factor de transcripción es una proteína que causa la activación de un promotor. Los factores de transcripción pueden ser represores o activadores. En caso de estar presente un factor de transcripción represor de un promotor concreto, el gen de este operón pasará a estar en estado *off*. En el caso en que esté presente una proteína que es Factor de Transcripción activador de un promotor, el gen de este operón pasará a estar en estado *on*.

Los operones constitutivos no necesitan la presencia de factores de transcripción para activarse, por lo tanto el gen de este operón permanecerá en estado *on* toda la simulación.

5.1.2. Puertas Lógicas

Los operones pueden estar regulados por la presencia de uno o dos factores de transcripción. Aquellos regulados por un único factor de transcripción sólo podrán ejecutar una puerta *yes*. Esta puerta podrá estar regulada por un factor de transcripción activador o represor. En caso de estar regulada por un activador el comportamiento es el de una puerta lógica *yes*. En caso de estar regulada por un represor, el funcionamiento

es el de una puerta *not*.

YES	
Input	Output
Activador	
1	1
0	0

Tabla 5.1: Puerta YES con activador.

YES	
Input	Output
Represor	
1	0
0	1

Tabla 5.2: Puerta YES con represor.

Los circuitos *and* y *or* están regulados por dos factores de transcripción. Estos dos circuitos, al igual que el circuito *yes*, pueden estar regulados por activadores y represores. En una misma puerta se pueden tener activadores y represores al mismo tiempo, ya que es posible que un factor de transcripción active un promotor y otro lo reprima.

AND		
Inputs		Output
Activador	Activador	
1	1	1
1	0	0
0	1	0
0	0	0

Tabla 5.3: Puerta AND con dos activadores.

AND		
Inputs		Output
Represor	Represor	
1	1	0
1	0	0
0	1	0
0	0	1

Tabla 5.4: Puerta AND con dos represores.

AND		
Inputs		Output
Represor	Activador	
1	1	0
1	0	0
0	1	1
0	0	0

Tabla 5.5: Puerta AND con activador y un represor.

En una puerta *and* que necesite dos factores de transcripción activadores, únicamente producirá una salida cuando ambos activadores estén presentes. Si esta puerta está condicionada por dos factores de transcripción represores, se producirá la salida siempre y cuando ambos represores no estén presentes. En caso de una combinación de activador y represor, la salida sólo se podrá dar cuando esté presente el activador pero no el represor.

OR		
Inputs		Output
Activador	Activador	
1	1	1
1	0	1
0	1	1
0	0	0

Tabla 5.6: Puerta OR con dos activadores.

OR		
Inputs		Output
Represor	Represor	
1	1	0
1	0	1
0	1	1
0	0	1

Tabla 5.7: Puerta OR con dos represores.

OR		
Inputs		Output
Represor	Activador	
1	1	1
1	0	0
0	1	1
0	0	0

Tabla 5.8: Puerta OR con un activador y un represor.

En una puerta *or* las salidas ocurren con mayor facilidad, ya que siempre que intervenga un activador éste tiene que estar presente independientemente del otro factor de transcripción. En caso de que esta puerta la regulen dos represores, no habrá salida cuando ambos estén presentes, si uno de los dos no está, se puede producir la transcripción del gen.

5.1.3. Activaciones/degradaciones de proteínas y ARNs

Una proteína cambia de un estado a otro cuando se dan las condiciones necesarias de la puerta lógica. En ese momento, la proteína no pasa directamente del estado *off* a *on*. Una proteína tarda un tiempo en madurar y en que haya concentración suficiente para que pueda activar algún otro promotor, este tiempo se conoce como tiempo de activación. De la misma forma, una proteína tarda un tiempo de degradación en dejar de tener una concentración suficiente y degradarse en el medio. Este otro tiempo se conoce como tiempo de degradación.

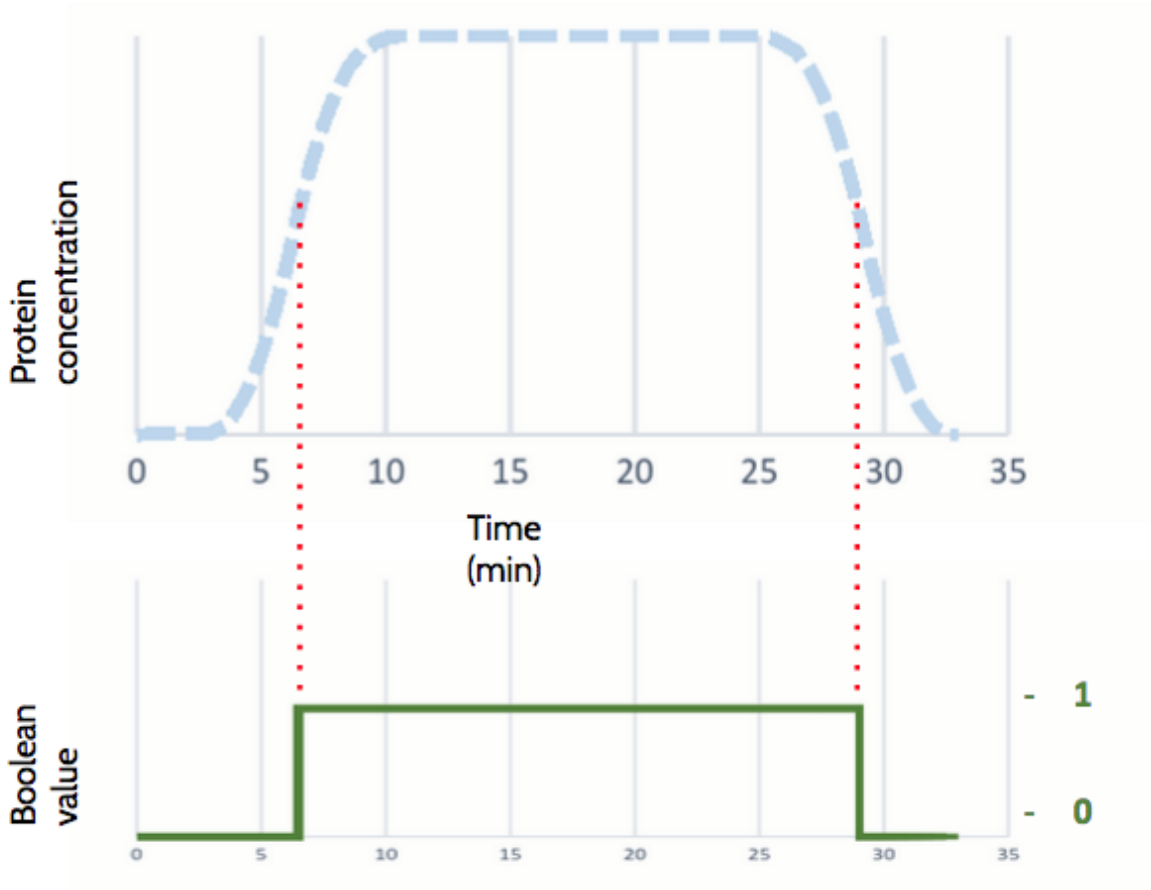


Figura 5.1: Simplificación de concentración de proteína

Una proteína tarda un tiempo t_a en llegar a su punto máximo de concentración. En

este modelo, se ha tenido en cuenta que la mitad de esa concentración ya es suficiente para que esa proteína active o reprima algún promotor, este tiempo se denomina *tiempo de semi-activación*. Es el tiempo que se espera desde que se dan las condiciones necesarias para que se active un gen hasta que se tiene la concentración de proteína suficiente para tomarla como *on*. La figura 5.1 muestra esta simplificación.

En degradaciones ocurre lo mismo que en activaciones. Una proteína tarda un tiempo t_d en degradarse por completo en el medio. Sin embargo, por debajo de la mitad de su concentración máxima no cuenta con el número de moléculas suficientes como para activar un promotor, por lo tanto no puede representarse como *on*. Por esta razón, una vez pasado el *tiempo de semi-degradación* de una proteína, ésta pasará del estado *on* al estado *off*.

Las activaciones y degradaciones de ARNs están modeladas con la misma dinámica que las proteínas. En el momento en que se den las condiciones necesarias para que la proteína asociada a un ARN concreto comience a activarse, el ARN también lo hará pero con un tiempo más corto que el tiempo de activación de la proteína, dado que las proteínas se sintetizan a partir del ARN, por lo tanto éste tiene que estar presente antes que la proteína. En caso de las degradaciones, el tiempo de degradación del ARN también será menor que el de degradación de la proteína.

5.1.4. Riboswitchs y moléculas inductoras

Un riboswitch posicionado delante de un gen bloquea la traducción de ARN. Este riboswitch sólo dejará que prosiga la traducción en presencia de una molécula de ARN concreta. En caso de tener un riboswitch delante de un gen, y no tener esa molécula de ARN que lo desbloquee, no se generará proteína ni ARN de los genes que están por detrás de ese riboswitch. En cambio, en operones con varios genes, los genes previos al riboswitch si podrán traducirse en ARN y posteriormente sintetizar proteína.

En el capitulo 2 se detalló que una molécula puede tanto inhibir la adherencia de un factor de transcripción a un promotor, como ser imprescindible para que el factor de transcripción se una al promotor. De la misma forma, se puede definir si es necesaria

la presencia de una molécula para que un promotor se active o si, por el contrario, la presencia de una molécula determinada hace que el promotor no pueda activarse.

5.1.5. Estado inicial

Otro de los parámetros que se pueden definir al iniciar un experimento, son las proteínas y los ARNs iniciales que va a tener una bacteria al comenzar la simulación. Las proteínas y los RNAs indicados estarán en estado *on* al comenzar la simulación sin la necesidad de esperar el tiempo de activación. Al iniciar la simulación, estas proteínas y RNAs comenzarán su tiempo de degradación siempre y cuando esa célula no contenga algún otro operon activado que genere esas proteínas. Ya que en ese caso, otro operón la genera y no se degradará.

Es interesante permitir definir proteínas iniciales, ya que permite comenzar un experimento desde un estado concreto, en lugar de esperar a que la propia dinámica defina la secuencia de estados. Por ejemplo, permite comenzar el Repressilator con la proteína TetR activada en lugar de esperar a la proteína que tarde un menor tiempo en activarse.

5.1.6. Ruido

El ruido en los modelos de simulación de agentes es un elemento bastante importante y poco común. Los modelos existentes de expresión genética no manejan el ruido aún dada su importancia. En el proceso de expresión genética pueden ocurrir mutaciones en los genes y casos en los que una proteína no se adhiera correctamente a un promotor. En estos casos, un gen que debiera activarse, no lo hará. Este modelo, contempla estas situaciones permitiendo al usuario definir un ruido determinado.

En cada operón puede definirse una probabilidad de éxito en las operaciones de activación/represión. Así, por ejemplo, es posible que un gen se active correctamente el 90 % de las veces en un operón determinado pero el otro 10 % no lo haga debido a mutaciones o fallos.

Adicionalmente, se añade ruido en los tiempos de activación/degradación. En el modelo, pude definirse una desviación típica de los tiempos de activación y degradación para

que no todos los genes iguales tarden lo mismo en activarse/degradarse. Además, en caso de tener dos genes con los mismos tiempos de activación/degradación siempre se activará primero el gen en el que se compruebe antes su estado. Para evitar que siempre se compruebe un gen antes que otro, se ha añadido aleatoriedad en las comprobaciones. Dentro de una misma bacteria, los operones se comprobarán de forma aleatoria para permitir que un gen no se active siempre antes que otro.

5.1.7. Actuadores

La salida de un circuito genético, además de la activación o represión de una proteína, puede producir algún efecto en la célula. Dependiendo de la proteína que esté activada se ejecuta una acción concreta en la bacteria. Entre estas acciones se encuentran las dinámicas de conjugación, brillo o muerte entre otras.

5.2. Diseño de circuitos genéticos

El diseño de los circuitos genéticos que van a intervenir en una simulación se realiza en lenguaje *gro*. Se ha ampliado este lenguaje para permitir el uso de nuevas funciones que definan ciertos aspectos de los nuevos circuitos. Cada característica de un circuito genético es parametrizable, evitando así cualquier suposición errónea en relación a las dinámicas de expresión genética. A continuación se detallarán las nuevas funciones que permiten el diseño de circuitos genéticos sintéticos.

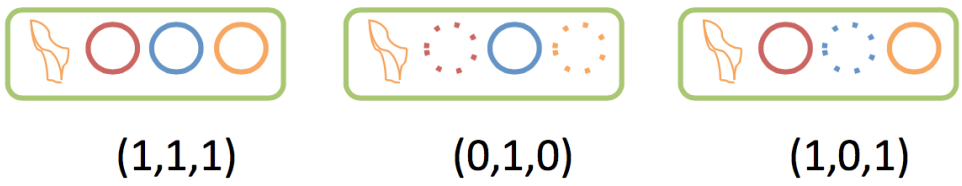


Figura 5.2: Lista de plásmidos contenidos en una célula

Cada bacteria puede tener una combinación diferente de plásmidos, o incluso ninguno. Los plásmidos que contiene cada bacteria pueden cambiar durante la simulación, ya puede ser porque la bacteria pierda un plásmido o porque adquiera algún otro por

conjugación. Esta combinación de plásmidos viene dada en cada bacteria según una lista booleana (fig. 5.2)

En el Apéndice A puede encontrarse un ejemplo completo de diseño de un circuito genético.

5.2.1. Parámetros

Lo primero es definir una serie de parámetros para inicializar las posteriores estructuras. Todos los siguientes parámetros se miden a nivel de colonia, por lo tanto se definirá el número de elementos que habrá a nivel global en toda la simulación.

- **Número de plásmidos** Número total de plásmidos en la colonia.
- **Número de proteínas** Número total de proteínas y ARNs en la colonia.
- **Número de riboswitchs** Número total de instancias de riboswitchs que se activan con diferentes ARNs en la colonia.
- **Número de moléculas** Número total de moléculas inductoras en la colonia.

```
set ("num_plasmids",2);
set ("num_proteins",4);
set ("num_riboswitchs",3);
set ("num_molecules",4);
```

5.2.2. Matriz de ARNs-riboswitchs

La matriz de ARNs-riboswitchs sirve para indicar los ARNs que activan cada riboswitch. Las columnas de la matriz indican el número de riboswitch y las filas los ARNs en el mismo orden que las proteínas a las que están asociados. Los ARN que activan cada riboswitch se definen mediante un valor booleano true/false. Varios ARNs pueden activar un único riboswitch, por lo que puede haber más de un valor *true* en la misma fila. La tabla 5.9 muestra un ejemplo de esta matriz.

	rbsw1	rbsw2	...	rbsN
RNA1	true	false	...	true
RNA2	false	false	...	true
RNA3	true	true	...	false
...	...	...	...	...
RNAM	false	false	...	true

Tabla 5.9: Matriz de RNAs que activan riboswitchs

Un ejemplo de la matriz de de ARNs-riboswitchs en *gro* en el que se definen cuatro riboswitchs con cuatro ARNs es el siguiente:

```
RNA_matrix({true, false, false, true,
            false, false, true, false,
            false, true, false, true,
            false, false, false, true});
```

En este ejemplo, el *riboswitch1* se activa con el *ARN1*. El *riboswitch2* se activa con el *ARN3*. El *riboswitch3* se activa con el *ARN2* y el *riboswitch4* se activa con los *ARN1, ARN3* y *ARN4*.

5.2.3. Matriz de moléculas inductoras

La matriz de moléculas inductoras indica las moléculas que inducen/inhiben alguna proteína. Se indica mediante un 1 las moléculas que inducen una proteína y mediante un -1 las que la inhiban. Las moléculas que no producen ningún efecto en una proteína se indican mediante un 0.

	prot1	prot2	...	protN
mol1	0	-1	...	0
mol2	0	1	...	0
mol3	1	0	...	0
...	...	...	...	...
molM	0	-1	...	0

Tabla 5.10: Matriz de moléculas que inducen/inhiben alguna proteína

A continuación se muestra un ejemplo de declaración de esta matriz. En el, la *molécula1* inhibe la *proteína2* e induce la *proteína3*, por lo que la *proteína2* no podrá comportarse como factor de transcripción cuando esta molécula este presente. Sin embargo, por el contrario, la *proteína2* solo podrá unirse al promotor de un operón cuando la *molécula1* esté presente. Por otro lado, la *molécula2* induce la *proteína3* y la *molécula3* induce la *proteína1*. Y finalmente, la *molécula4* inhibe la *proteína2*.

```
molecules_matrix({0,-1,1,0,
                  0,0,1,0,
                  1,0,0,0,
                  0,-1,0,0});
```

5.2.4. Operones

Los operones son una estructura básica en este módulo. En ellos vienen definidos los genes que los conforman así como todos los parámetros que los definen. Un operón viene definido por los siguientes parámetros:

- **Lista de proteínas de salida** Lista con valores booleanos indicando qué proteína se va a activar/reprimir cuando se den las condiciones de la puerta lógica. Se activarán también los ARNs asociados a esas proteínas.
- **Lista de ARNs de salida** Lista con valores booleanos indicando qué ARNs se va a activar/reprimir cuando se den las condiciones de la puerta lógica. No es necesario indicar los ARNs asociados a las proteínas de salida indicadas en la

lista de proteínas. La función de esta lista es representar operones que únicamente traduzcan el ADN en ARN pero no transcriban ese ARN en proteína.

- **Lista de Factores de Transcripción** Lista con posibles valores {0,1,-1} indicando mediante un 0 que esa proteína no es factor de transcripción de este operón. Mediante un 1 se representa a una proteína que es factor de transcripción inductor de este operón, es decir, si esa proteína está presente las proteínas de salida comenzarán a activarse. Y por último, mediante un -1 se representa a un factor de transcripción represor, es decir, en presencia de este factor de transcripción, las proteínas de salida se reprimirán.

- **Lista de Riboswitchs** Lista con valores {0 ... num_riboswitchs}. Se utiliza el número de riboswitch según el orden indicado en la matriz de riboswitchs, y se indica en la posición de lista delante del gen donde se pretende insertar el riboswitch. Por ejemplo, si la lista es {0,1,0,4} representa que el *riboswitch1* está delante del *gen2* y que el *riboswitch4* está delante del *gen4*.

- **Constitutivo** Valor booleano indicando con *true* que el operón es constitutivo y con *false* que no lo es. Si el operón es constitutivo no es necesario indicar ningún factor de transcripción ya que no son necesarios en operones constitutivos, en ese caso la lista de factores de transcripción será una lista de ceros.

- **Tiempos de activación de las proteínas** Lista de valores decimales en la que se indica el tiempo de semi-activación de las proteínas.

- **Desviación típica de los tiempos de activación de las proteínas** Lista con valores decimales en la que se indica la desviación típica de los tiempos de semi-activación de las proteínas.

- **Tiempos de activación de los ARNs** Lista de valores decimales en la que se indica el tiempo de semi-activación de los ARNs.

- **Desviación típica de los tiempos de activación de los ARNs** Lista con valores decimales en la que se indica la desviación típica de los tiempos de semi-activación de los ARNs.

- **Circuito** Valor entre {0,1,2} indicando la puerta lógica que posee el promotor de este operón. 0 representa a la puerta *yes*, 1 representa a la puerta *and* y 2 representa a la puerta *or*.

- **Probabilidades para activaciones**
  $P_{activador}(output0/input0)$: probabilidad de que no se exprese proteína dado que no hay activador
  $P_{activador}(output0/input1)$: probabilidad de que no se exprese proteína dado que hay activador
  $P_{activador}(output1/input0)$: probabilidad de que se exprese proteína dado que no hay activador
  $P_{activador}(output1/input1)$: probabilidad de que se exprese proteína dado que hay activador

- **Probabilidades para represiones**
  $P_{represor}(output0/input0)$: probabilidad de que no se exprese proteína dado que no hay represor
  $P_{represor}(output0/input1)$: probabilidad de que no se exprese proteína dado que hay represor
  $P_{represor}(output1/input0)$: probabilidad de que se exprese proteína dado que no hay represor
  $P_{represor}(output1/input1)$: probabilidad de que se exprese proteína dado que hay represor

Las probabilidades en activaciones y represiones son diferentes porque no es seguro que las afirmaciones $P_{activador}(output1/input1) = P_{represor}(output1/input0)$ y $P_{activador}(output0/input0) = P_{represor}(output0/input1)$ sean correctas. El hecho de que ocurra con la misma probabilidad que se exprese proteína en presencia de un activador que en ausencia de un represor no es una afirmación demostrada. Al igual que la afirmación de que ocurra con la misma probabilidad que no se exprese proteína en ausencia de un activador y en presencia de un represor. Por esta razón, se deja la elección al usuario permitiendo definir las probabilidades en cada caso.

Estas probabilidades se ejecutan por ciclo de vida de la célula. Es decir, si $P_{activador}(output1/input0) = 0{,}1$, entonces en un ciclo de vida, el hecho de que una proteína se

active no teniendo activador ocurrirá el 10 % de las veces.

A continuación se muestra un ejemplo de definición de un operón. En él, se define la *proteína2* como proteína de salida. En este caso no hay ningún ARN que se vaya a traducir sin transcribirse, únicamente se va a traducir el *ARN2* y posteriormente sintetizarse en la *proteína2* por ser la proteína de salida. La *proteína3* actúa como factor de transcripción inductor, por tanto en presencia de esta proteína se activará la expresión del *gen2* para producir *proteína2*. Sin embargo, el *gen2* tiene al *riboswitch2* delante bloqueando su traducción. No es un operón constitutivo. El tiempo de activación de la *proteína2* es de 10,0±2,3 min. El tiempo de activación del *ARN2* es de 3,0±0,3 min. La puerta lógica del promotor es una puerta *yes*. Debido a que este operón no tiene ningún factor de transcripción represor, solo se indicarán las propiedades de activadores.
$P_{activador}(output0/input0) = 0{,}9$
$P_{activador}(output0/input1) = 0{,}2$
$P_{activador}(output1/input0) = 0{,}1$
$P_{activador}(output1/input1) = 0{,}8$

```
operon ({false,true,false,false}, {false,false,false,false},
        {0,0,1,0}, {0,2,0,0}, false,
        {0.0,10.0,0.0,0.0}, {0.0,2.3,0.0,0.0},
        {0.0,3.0,0.0,0.0}, {0.0,0.3,0.0,0.0},
        0, {0.9,0.2,0.1,0.8}, {0,0,0,0});
```

5.2.5. Tiempos de degradación

Los tiempos de degradación, a diferencia de los tiempos de activación, no dependen del promotor que haya sido activado para generar la proteína. Cada proteína tiene su propio tiempo de degradación definido globalmente en el experimento. Las primeras dos listas de la función corresponden a los tiempos de degradación de las proteínas y sus desviaciones típicas y, las dos siguientes listas corresponden a los tiempos de degradación de los ARNs y sus correspondientes desviaciones típicas.

A continuación, se muestra un ejemplo de esta función:

```
degradation_times({10.0,5.0,5.0,2.0},{1.2,2.3,1.4,1.2},
                  {2.0,3.0,2.0,4.0},{0.2,0.3,0.4,0.2});
```

5.2.6. Matriz de plásmidos

Una estructura imprescindible en el diseño de un experimento en este módulo es la Matriz de Plásmidos. En esta matriz, en la que las columnas son operones y las filas plásmidos, se definen los operones que conforman cada plásmido. En la tabla 5.11 se muestra un ejemplo de esta matriz. Los operones que forman los plásmidos se definen mediante un valor booleano true/false.

	op1	op2	...	opN
plas1	true	false	...	false
plas2	false	true	...	true
plas3	true	false	...	false
...	...	...	...	...
plasM	true	false	...	true

Tabla 5.11: Matriz de plásmidos

En *gro* la matriz de plásmidos se define de la siguiente forma:

```
plasmids_matrix ({true,true,true,true,
                  false,true,false,false});
```

En este ejemplo, el *plásmido1* está formado por el *operón1, operón2, operón3* y *operón4*, y el *plásmido2* está formado únicamente por el *operón2*.

5.2.7. Acciones

Dependiendo del estado de las proteínas en una célula se ejecutarán ciertas acciones. Por ejemplo, si una célula tiene concentración suficiente de la proteína GFP, esta bacteria deberá brillar con luz verde fluorescente. Estas acciones se contemplan en el modelo mediante la función *action*. El primer parámetro que se le pasa a esta función es una lista con las proteínas que causan esa acción. La acción se ejecutará cuando las proteínas indicadas como *true* estén en estado *on* independientemente del estado de las proteínas indicadas como *false*. El segundo parámetro es el nombre de la acción a realizar, por ahora solo hay disponibles tres acciones aunque posteriormente se pueden ampliar. Estas acciones son pintar (*paint*), conjugar (*conjugate*) y morir (*die*). El tercer argumento que recibe la función action son los parámetros que van a llevar cada una de las acciones.

La función *paint* sirve para cambiar el color de la bacteria. Los parámetros que va a tener esta función son las concentraciones del nivel de proteínas que causan los colores verde, rojo, amarillo y azul, {GFP, RFP, YFP, BFP}. Estos valores variarán entre 0 como concentración nula y 100 como concentración máxima de proteína.

```
action({false,true,false},"paint",{"100","0","0","0"});
action({false,false,true},"paint",{"0","100","0","0"});
action({true,true,false},"paint",{"50","50","0","0"});
action({true,false,true},"paint",{"0","100","0","100"});
```

La función *conjugate* conjugará un plásmido concreto con una tasa de conjugación determinada. El plásmido a conjugar será el primer parámetro que ha de pasarse a la función y la cantidad de conjugaciones en un ciclo de vida es el segundo.

```
action({false,true,false},"conjugate",{"1","0.2"});
action({true,false,false},"conjugate",{"2","0.1"});
```

La función *die* provoca la muerte a una célula en un tiempo determinado. El único argumento necesario para esta acción es el número de minutos que han de pasar para provocar la muerte a la bacteria.

```
action({true,false,true},"die",{"10"});
action({true,false,true},"die",{"20"});
```

5.2.8. E. Coli

Debido al nuevo módulo genético es necesario indicar nuevos parámetros al crear las células para comenzar una simulación. Además de los parámetros que ya eran necesarios antes de esta ampliación del lenguaje *gro*, se han añadido tres nuevas listas con valores booleanos para indicar los plásmidos contenidos en esta célula y las proteínas y ARNs iniciales que posee al iniciar la simulación. En el siguiente ejemplo, esa célula contiene el primer plásmido, la proteína2 y el ARN2.

```
ecoli ( [ x := 0, y := 0], {true, false},
        {false,true,false,false}, {false,true,false,false},
        program p() );
```

5.2.9. Agregar molécula inductora

Las moléculas inductoras suelen agregarse de forma externa a la colonia. Por esta razón, estas moléculas se definen a nivel global con valores digitales, cuando se agregue molécula se representa con un (1) y en su ausencia (0). Cuando una molécula está presente, está presente para todas las células de la colonia. Para agregar una molécula al medio se ha definido la función set_molecule. Esta función cambia el estado de una molécula de *on* a *off* o viceversa. A continuación se muestra un ejemplo de la función set_molecule.

```
set_molecule(1,1);
set_molecule(3,0);
```

En el ejemplo, se modifica el estado de la *molécula1* a *on* y el estado de la *molécula3* a *off*.

5.3. Procedimiento

La regulación genética es un procedimiento complejo que necesita comprobaciones periódicas de estados de proteínas. La comprobación de estos estados es un proceso computacionalmente costoso que aumenta según va creciendo la colonia. Debido a estos costes, es necesario realizar un modelo simplificado que permita simular este proceso en una colonia de al menos 10^5 bacterias.

Una vez inicializadas todas las estructuras necesarias que se detallan en el apartado anterior. El primer paso consiste en comprobar los plásmidos que contiene una bacteria. Según los plásmidos que contenga, se comprueban los operones que conforman esos plásmidos según la matriz de plásmidos. En caso de que la bacteria contenga varios plásmidos diferentes que compartan algún operón, ese operón se comprobará solo una vez. Dado que, si en una misma bacteria hay dos operones iguales, ambos comenzarán la expresión de sus genes al mismo tiempo porque se cumplirán las mismas condiciones al pertenecer a la misma célula. Para evitar realizar los mismos cálculos dos veces por bacteria, se comprueba cada operón una única vez en cada iteración. Siempre y cuando ambos comiencen su expresión al mismo tiempo, ya que en caso de que se acabe de conjugar un plásmido con un operón que ya se estuviera activando en la colonia, se mantendría el tiempo del operón que se estaba activando y no se tendría en cuenta el operón que acaba de conjugarse.

Los operones se analizan de forma aleatoria para evitar que en caso de que dos genes se activen a la vez, uno de los dos siempre se active siempre antes que el otro. Este proceso añade estocasticidad al método, ya que no es posible predecir cual de los dos genes se activará primero.

En cada operón, el primer paso es comprobar si es constitutivo. En operones constitutivos, el RNA y la proteína correspondientes a los genes de esos operones comienzan a activarse inmediatamente, ya que no necesitan la presencia de factores de transcripción para iniciar su expresión. Los operones no constitutivos, necesitan comprobar la presencia de sus factores de transcripción y, dependiendo del circuito (*yes, and, or*), se comprueban una serie de reglas lógicas para decidir si un gen debe comenzar a expre-

sarse. Las reglas son del tipo:

*"Si la proteina **x** es activador, tengo proteina **x** y antes no tenía, entonces activar las proteínas y los RNAs de salida"*

*"Si la proteina **x** y la proteina **y** son activadores, tengo proteina **x** y proteína **y** y antes no tenía proteina **x** ni proteína **y**, entonces activar las proteínas y los RNAs de salida"*

*"Si la proteina **x** es represor, tengo proteina **x** y antes no tenía, entonces degradar las proteínas y los RNAs de salida"*

Según estas reglas, se aprecia que es necesario conocer el estado anterior de las proteínas en la bacteria, ya que comparando el estado anterior con el estado actual se puede determinar si ha ocurrido un cambio de estado. En cada iteración se guarda el estado de las proteínas y RNAs para poder consultarlo en la siguiente iteración y saber así si alguna proteína/RNA se ha cambiado de estado.

En el momento en que se dan las condiciones necesarias para que una proteína/RNA cambie de estado, se consulta su tiempo de activación o degradación, según el cambio que se esté realizando.

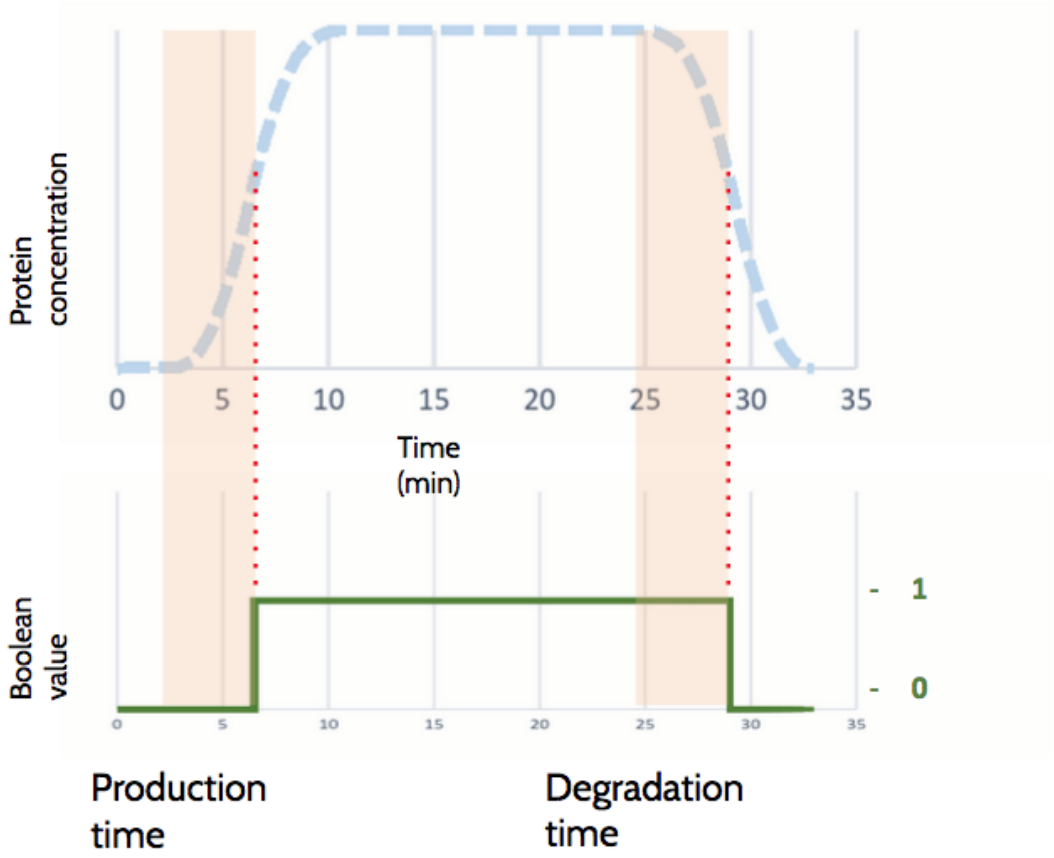


Figura 5.3: Tiempo de activación/degradación en inducción

En inducciones, se tiene en un primer momento la proteína a *off*. Se calcula el tiempo que va a necesitar la proteína para activarse añadiendo al minuto de simulación actual el tiempo de activación indicado en el diseño del circuito más la desviación típica. En las siguientes iteraciones se comprobará si se ha alcanzado ese tiempo. En el momento en que ese tiempo se alcance, se cambia de estado a la proteína a *on*. Esta proteína se mantendrá en este estado siempre y cuando el factor de transcripción que hace que esta proteína se exprese siga presente. Si este factor de transcripción deja de estar presente, se realizará el mismo proceso a la inversa. Se esperará un tiempo de degradación y se cambiará el estado de la proteína a *off* (fig. 5.3).

En represiones, el proceso es el contrario. En ausencia de factor de transcripción repre-

sor la proteína de salida se mantiene en estado *on*. En el momento en que se comprueba que hay factor de transcripción, se guarda el tiempo de degradación que va a tardar la proteína en degradarse. Una vez esperado ese tiempo, la proteína cambia de estado, al estado *off*. Esta proteína sólo cambiará de estado cuando deje de haber factor de transcripción represor. En ese momento se esperará el tiempo de activación y se cambiará su estado de nuevo a *on*.

En la división de la bacteria, suponemos que la concentración de proteínas que tenía la madre se divide a la mitad cuando se forman las dos hijas, quedándose cada una con su respectiva mitad. Se ha considerado que la mitad de la concentración de la madre no es suficiente para considerarla un *on* en el modelo, ya que eso supondría que una bacteria que acaba de dividirse puede comenzar a ejecutar acciones como conjugar o brillar. Normalmente, estas bacterias necesitan un tiempo de maduración para conseguir llegar de nuevo a cantidades suficientes de concentración de proteínas. Para modelar este proceso, las proteínas que estaban en estado *on* en la madre, pasan a estar en estado *off* en las hijas y esperan la mitad del tiempo de activación predefinido para volver a estar en estado *on*.

Justo antes de comenzar a ejecutar las reglas lógicas, se mira si alguno de los genes tiene un riboswitch delante. En ese caso, se busca en la matriz de riboswitchs qué RNA activa este riboswitch. Si ese RNA está presente en la célula, entonces se procede con la expresión. En caso contrario se interrumpe y no podrá comenzar a activarse hasta que el RNA esté presente.

En presencia de moléculas inductoras, hay que tener en cuenta que antes de que un factor de transcripción se una al promotor hay que fijarse si hay alguna molécula inductora que evita que esto ocurra. En caso de que necesite la presencia de una molécula concreta para que el factor de transcripción pueda adherirse al promotor, no lo hará hasta que esta molécula esté presente. Por otro lado, en caso de que una molécula inhiba la adherencia, el factor de transcripción se unirá siempre y cuando esta molécula no esté presente.

A continuación se muestra un pseudocódigo que resume el proceso. En él no se mues-

tran las reglas lógicas que se ejecutan en cada puerta, únicamente las comprobaciones a realizar antes de proceder al paso de lógica.

Pseudocódigo

```
procedimiento comprobar_operon()
  Desde i =0 hasta i=num_plasmidos-1 hacer
       Desde j =0 hasta j=num_operones-1 hacer
            Si matriz_plasmidos[i][j] = true
                 guardar operón para luego comprobar aleatoriamente
                 j=num_operones
fin procedimiento comprobar_operon


procedimiento actualizar_proteinas()
  Si es constitutivo
       desde i =0 hasta i = num_proteinas-1 hacer
            si la proteina i es proteina de salida de este operon
                 si la proteina i no se está activando y no esta en estado on
                       activar proteína i
                  en caso contrario
                       comprobar si ha alcanzado el tiempo de activación

  Si circuito = 0 // Puerta YES
       desde i =0 hasta i = num_proteinas-1 hacer
            si la proteina i es factor de transcripción de este operon
                 desde j=0 hasta j = num_proteinas-1 hacer
                     si la proteina j es proteina de salida de este operon
                          si no hay ningún riboswitch delante del gen o
                               hay RNA que active el riboswitch
                                    comprobar reglas lógicas para la puerta YES

  Si circuito = 1 o circuito = 2 // Puertas AND /OR
       desde i =0 hasta i = num_proteinas-1 hacer
            si la proteina i es factor de transcripción de este operon
                 guardar y buscar el siguiente factor de transcripción
       desde j=0 hasta j = num_proteinas-1 hacer
            si la proteina j es proteina de salida de este operon
                  si no hay ningún riboswitch delante del gen o
                  hay RNA que active el riboswitch
                        si circuito = 1
                              comprobar reglas lógicas para la puerta AND
                        si circuito = 2
                              comprobar reglas lógicas para la puerta OR
fin procedimiento actualizar_proteinas
```

Capítulo 6

Resultados y Validación

6.1 Repressilator 68
6.2 Ruido 75
6.3 Supervivencia de plásmidos 78

Para verificar que el nuevo módulo es capaz de simular procesos biológicos, se ha escogido el circuito oscilatorio *Repressilator* para recrear su comportamiento en el simulador y posteriormente comparar los resultados con los obtenidos experimentalmente según el artículo *A synthetic oscillatory network of transcriptional regulators* [16].

Dado que el ruido es un elemento novedoso en el diseño de modelos de regulación genética, es importante verificar su correcto funcionamiento. Para ello se ha simulado un circuito sencillo en el que el promotor siempre está reprimido pero se activa espontáneamente por ruido.

Y por último, además de verificar el correcto comportamiento del modelo, se propone una aplicación. Gracias a este nuevo módulo es posible probar circuitos con diferentes parámetros y ver que propiedades se tienen que cumplir en ellos para dar un resultado determinado. En este caso, se ha intentado averiguar cual debe ser la tasa mínima de conjugación de un plásmido para que éste asegure su supervivencia en la colonia.

6.1. Repressilator

El circuito *Repressilator* es un circuito formado por dos plásmidos. En el primer plásmido se codifican tres genes TetR, LacI y cI. Cada uno de estos tres genes es factor de transcripción represor del siguiente gen en el ciclo, tal y como se muestra en el plásmido de la izquierda de la figura 6.1. El segundo de los plásmidos contiene un único gen que codifica la proteína GFP (plásmido de la derecha en la figura 6.1). Esta proteína está reprimida por el factor de transcripción TetR. Por tanto, la célula que contenga ambos plásmidos brillará únicamente cuando la proteína TetR se encuentre en estado *off*.

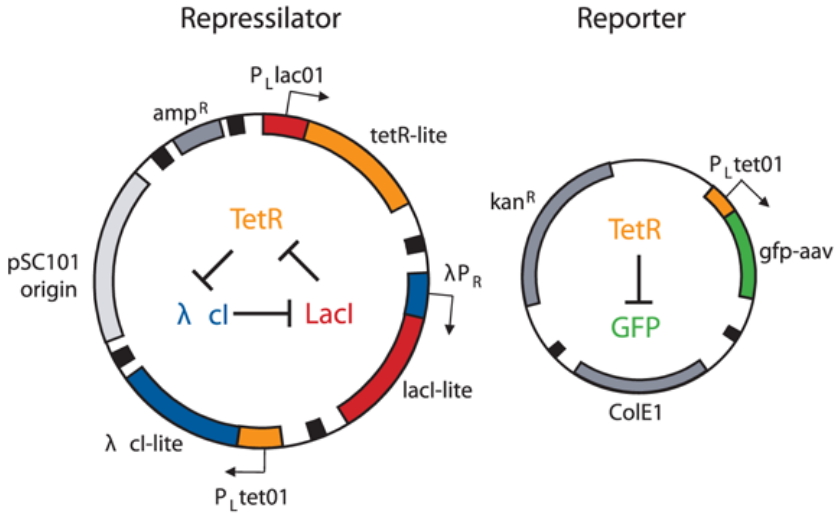


Figura 6.1: Circuito genético del Repressilator

El comportamiento esperado de este circuito consiste en que cuando una de las proteínas se active, la proteína a la que reprime comenzará a degradarse. A causa de la degradación de esta proteína, la tercera proteína, a la que reprime la segunda, comenzará a activarse causando la degradación de la primera. Esta dinámica produce un comportamiento oscilatorio entre los tres genes que componen el *Repressilator*.

Dado que existen dos modelos con los que representar las dinámicas de expresión genética (modelo basado en EDOs y modelo booleano), la figura 6.2 representa los estados del Repressilator según ambos modelos. En el modelo basado en EDOs, se observa una

dinámica mas suavizada permitiendo que un gen comience a activarse mientras otro se degrada. Al contrario que en el modelo booleano en el que se debe esperar a que una proteína cambie al estado *off* para que otra cambie al estado *on*. En el modelo booleano de la figura no se permite la expresión de más de un gen al mismo tiempo, caso que ocurre en casos reales o en el modelo de EDOs, ya que mientras uno se degrada otro se está activando, y dependiendo del tiempo que tarde cada uno de ellos en degradarse y en activarse respectivamente, es posible que más de un gen esté activado al mismo tiempo.

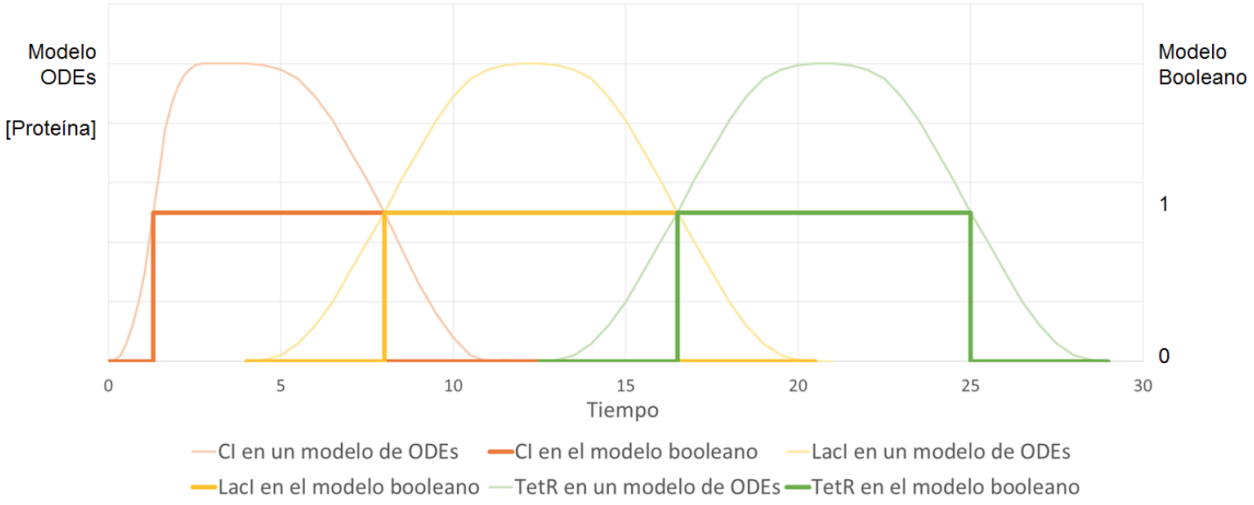


Figura 6.2: Repressilator modelo

Este modelo booleano de expresión genética trabaja con retrasos en los tiempos de activación y degradación de las bacterias. Este retraso permite que dos genes estén activados al mismo tiempo ya que en el momento en el que un factor de transcripción represor se ha activado se espera un tiempo hasta que se degrada la proteína a la que reprime, por lo tanto, en ese tiempo se tienen ambas proteínas activadas.

En el artículo [16] se muestran resultados sobre la expresión media de la GFP en una única célula a la que se le ha introducido el circuito *Repressilator*. Estos datos se muestran en la figura 6.3. Con el fin de validar el comportamiento de este modelo probabilístico con retrasos, se ha diseñado un circuito *Repressilator* en lenguaje *gro* con los parámetros que detallan en el artículo para poder comparar ambos resultados.

Este circuito puede encontrarse en el Apéndice B. Este experimento comienza con una colonia de 30 bacterias, todas ellas teniendo todos sus genes en estado *off*. Se ha dejado ejecutar hasta que ha alcanzado un tiempo de simulación de 600min para comparar con los resultados experimentales.

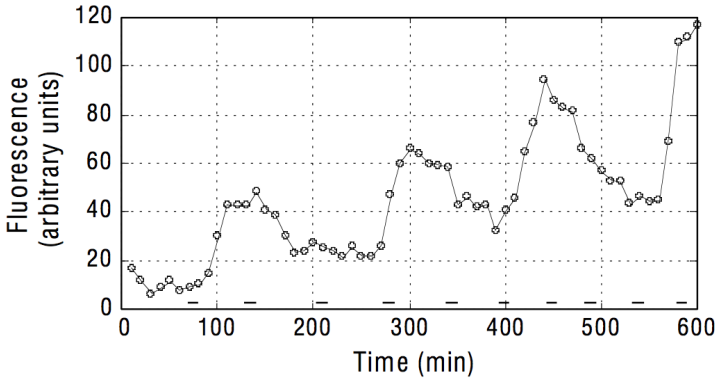


Figura 6.3: Expresión de la GFP en una célula real en la que se ha introducido el circuito Repressilator [16]

En la figura 6.3, se pueden observar ciclos de unos 150min, esto quiere decir que cada 150min la concentración de GFP alcanza un máximo relativo. Dado que en un modelo booleano no es posible conocer las concentraciones de proteínas, ya que solo tienen estado *on* y *off*, los resultados se representarán por medio de funciones booleanas.

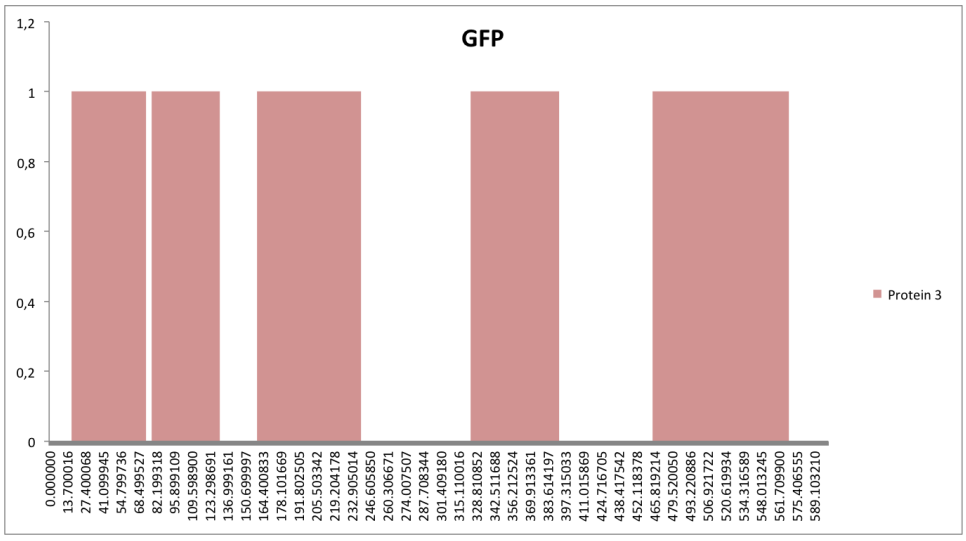


Figura 6.4: Estado booleano de la proteína GFP en una simulación del *Repressilator*

La figura 6.4 muestra el estado de la proteína GFP en una única célula con el circuito *Repressilator*. En ella, se observan ciclos de aproximadamente 150min muy similares a los obtenidos experimentalmente. Además, ocurren en tiempos de simulación muy próximos. El primer máximo ocurre cercano a los 100min de simulación, el segundo ronda los 250min, el tercero en torno a los 400min y el cuarto al rededor de los 550min.

El modelo booleano no posee la capacidad de representar concentraciones de proteínas, pero el simulador proporciona a cada bacteria una variable denominada *gfp* que viene representada por un número entero. Esta variable se utiliza para cambiar el color de la bacteria y que se represente con mayor o menor cantidad de color verde. Se ha usado esta variable para poder realizar una aproximación de las concentraciones de proteína GFP. En cada iteración, si la proteína GFP está en estado *on* se suma 20 al valor de la variable *gfp*, y si está en estado *off* se resta 20. De esta forma, se han representado estás "concentraciones" y se ha obtenido el gráfico de la figura 6.5. En la figura se aprecia que a medida que avanza la simulación, las concentraciones de proteína aumentan, de la misma forma que ocurre en la figura 6.3. Dado que las concentraciones en la simulación dependen únicamente del tiempo que el gen que produce GFP está activo, el hecho de que se obtengan resultados similares induce a que el tiempo que el gen está activado coincide también con el tiempo en que el gen está activado en los resultados del laboratorio.

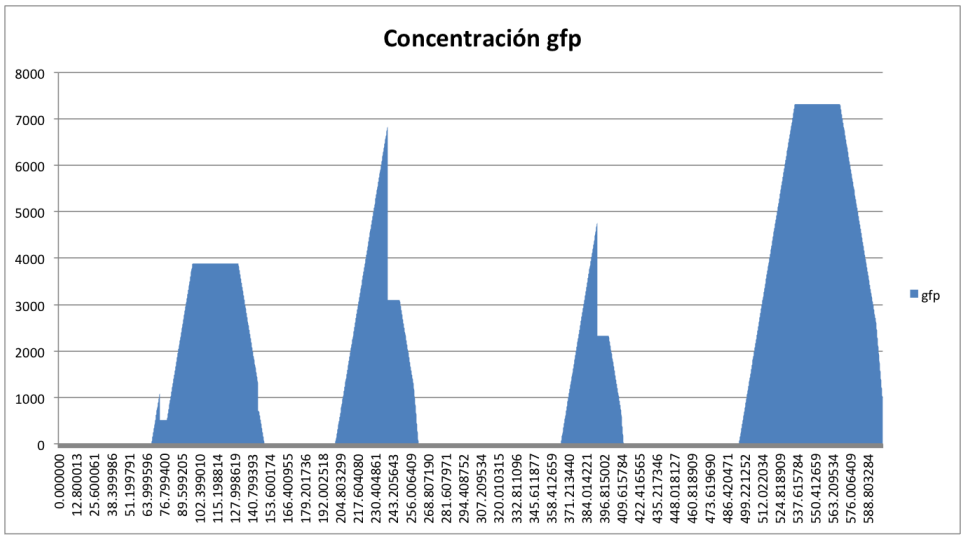


Figura 6.5: Concentración de GFP en una simulación del *Repressilator*

En el siguiente gráfico (fig. 6.6) se ha realizado una media con los datos obtenidos de la variable *gfp* en 10 simulaciones. En él se aprecia una dinámica muy similar a la del gráfico 6.3 presentando ciclos semejantes en torno a los instantes de tiempos en los que ocurren los ciclos de los datos experimentales.

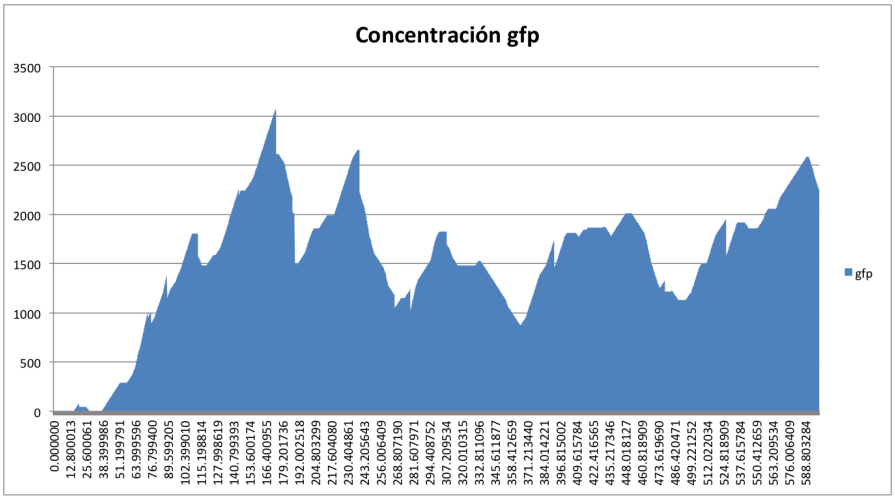


Figura 6.6: Concentración media de GFP obtenida en 10 simulaciones del *Repressilator*

Las figuras 6.7, 6.8 y 6.9 muestran los estados booleanos de las proteínas TetR, LacI y cI respectivamente en una simulación. En todas ellas se aprecian ciclos en torno a los 150min, no siendo todos ellos regulares debido al ruido que haya podido surgir.

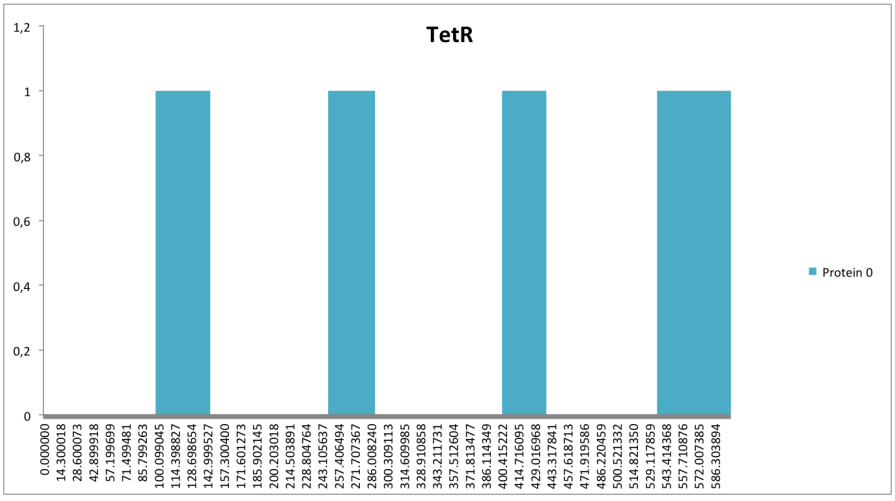


Figura 6.7: Estado booleano de la proteína TetR en una simulación del *Repressilator*

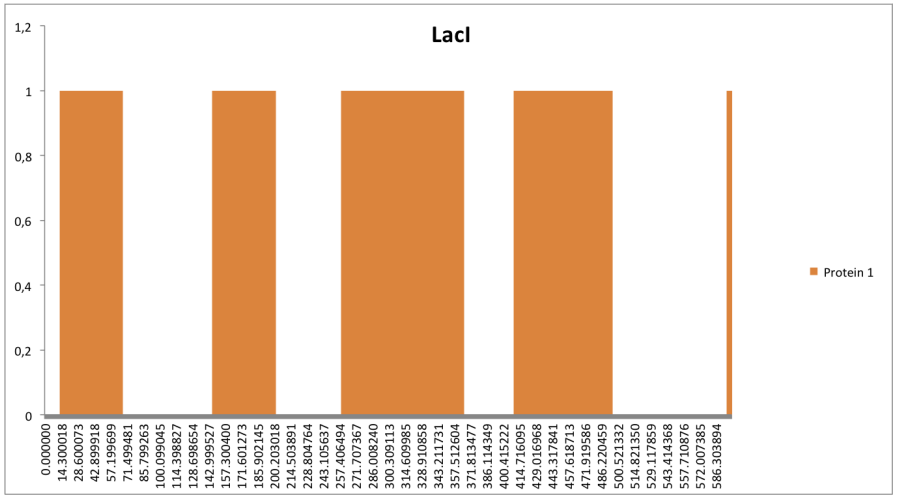


Figura 6.8: Estado booleano de la proteína LacI en una simulación del *Repressilator*

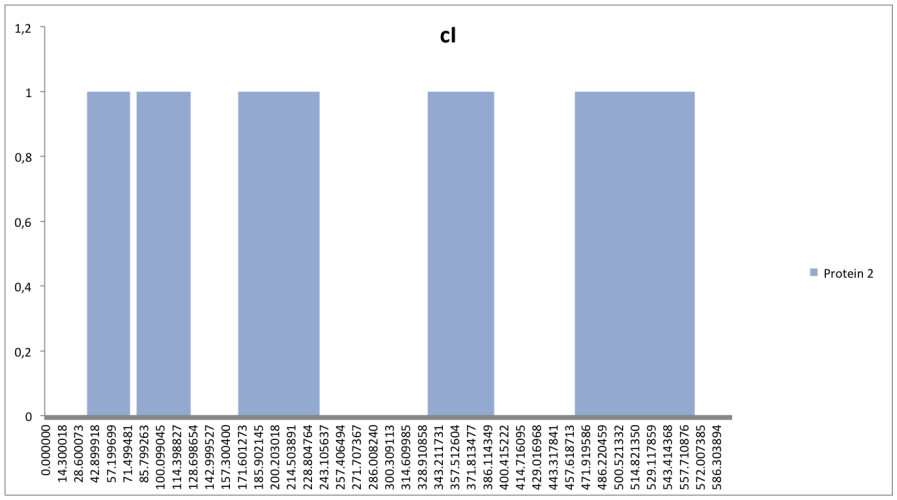


Figura 6.9: Estado booleano de la proteína cI en una simulación del *Repressilator*

La figura 6.10 representa los tres gráficos anteriores solapados. Las partes en las que aparece una barra vertical con la mitad de un color y la otra mitad de otro color representan que en ese momento dos proteínas están en estado *on* al mismo tiempo. También se observan casos en los que ninguna de las tres proteínas está activada, esto se debe a que a una de las proteínas se acaba de degradar y las otras dos aún están esperando a que termine su tiempo de activación. A excepción de estos casos, se aprecia un patrón entre las proteínas que se encuentran activadas en la mayoría de los casos,

debido al ruido hay veces que no siguen el patrón y se observa una variación. Por ejemplo, siempre que la *proteína0* está activada la siguiente proteína que se activa es la *proteína1*. Cuando la *proteína1* está activada la siguiente que se activa es la *proteína2*. Y cuando la *proteína2* está activada la siguiente que se activa es la *proteína0*. Por tanto, presenta una dinámica oscilante entre las tres proteínas respetando los ciclos y con casos de ruido.

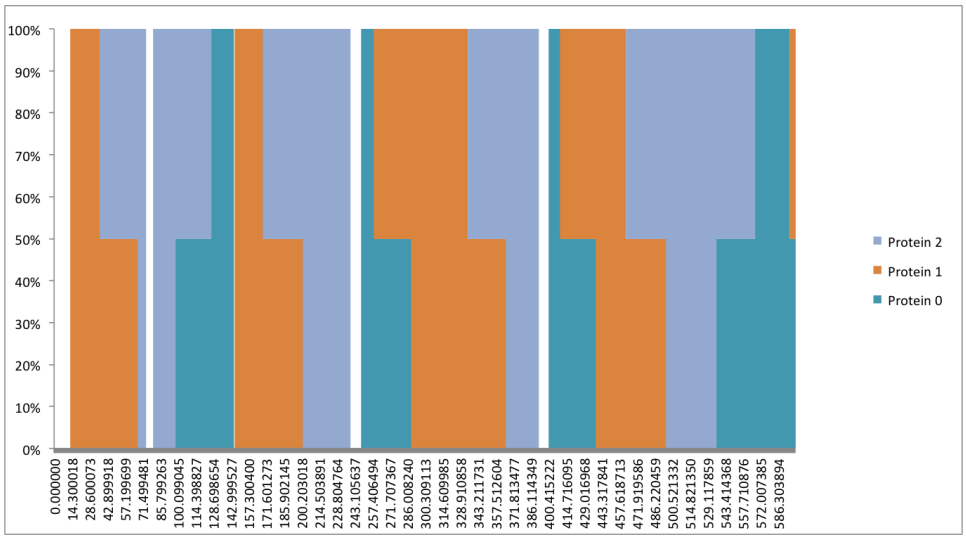


Figura 6.10: Estados booleanos de las proteínas TetR, LacI y cI en una simulación del *Repressilator*

Además de recopilar información de una única bacteria, se han obtenido datos de la colonia. En la simulación, se ha recogido la cantidad de células que tienen una proteína determinada en estado *on*. La figura 6.11 representa estos datos. En ella se aprecia igualmente una dinámica oscilante entre la cantidad de bacterias que tienen una determinada proteína. Esto se debe a que al comenzar la simulación todas las bacterias de la colonia tienen todos sus genes en estado *off*, por lo tanto, por la propia dinámica, la mayor parte de las células van a lograr una sincronización en sus ciclos activándose en ellas las mismos genes en instantes similares. La cantidad de células con proteínas que son represoras unas de otras presentan el mismo comportamiento oscilante propio del *Repressilator*, además las células con *proteína3* y *proteína4* que corresponden a las proteínas cI y GFP respectivamente, presentan el mismo ciclo, ya que por el propio diseño del circuito ambas proteínas se expresan al mismo tiempo ya que a ambas las

reprime la proteína TetR.

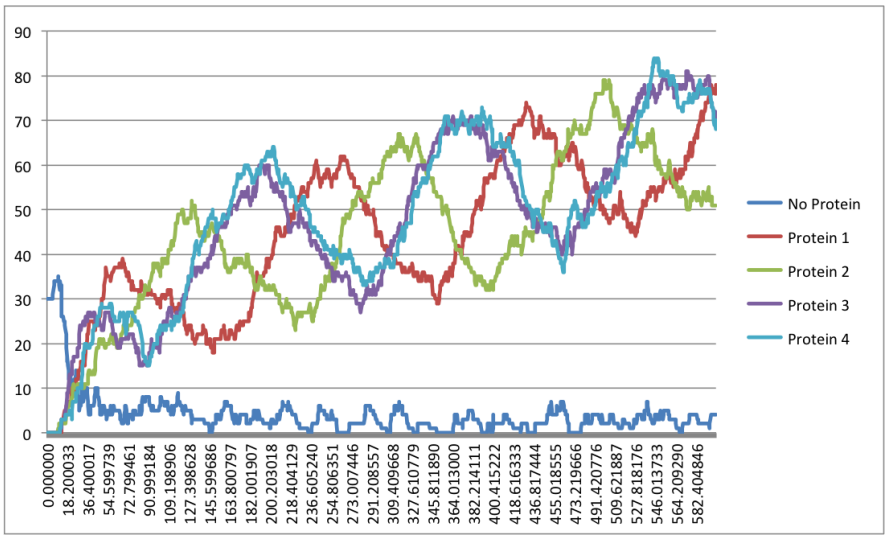


Figura 6.11: Cantidad de células que tienen cada una de la proteínas en estado *on* en una simulación del *Repressilator*

Por tanto, se ha demostrado que los resultados obtenidos con el simulador son muy similares a los obtenidos experimentalmente, respetando ciclos e instantes en los que los genes se activan. Además, se simulan casos de mutaciones en genes o fallos en la síntesis de proteína con ruido, consiguiendo con ello un comportamiento estocástico que igualmente cumple con las dinámicas del *Repressilator* experimental.

6.2. Ruido

Con el fin de validar el correcto funcionamiento del ruido, se ha diseñado un circuito en el que se utiliza el brillo que produce la proteína GFP como testigo de que un gen se ha activado por ruido. En el circuito se tiene un promotor que va a permanecer reprimido durante toda la simulación, ya que cada una de las células va a tener en estado *on* a la proteína que reprime este promotor. Este promotor regula la expresión de proteína GFP, por lo tanto, dadas las condiciones, la célula nunca brillaría en ausencia de ruido. La figura 6.12 muestra el efecto del ruido en este circuito. Para verificar que los datos insertados al diseñar el experimento coinciden con los datos obtenidos al ejecutarlo, se

han añadido diferentes probabilidades de ruido al mismo circuito y comprobado si la cantidad de bacterias que brillan debido al ruido es la correcta. El diseño del circuito usado en esta prueba puede encontrarse en el Apéndice C.

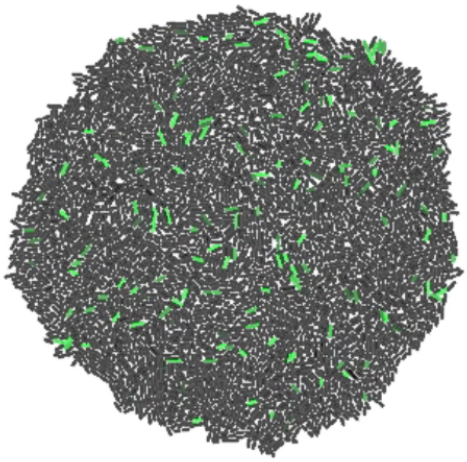


Figura 6.12: Ruido en una colonia

Para verificar el funcionamiento del ruido, se han extraído datos sobre la cantidad de bacterias que se activan por ruido. Las pruebas muestran la cantidad de células activadas (*n_glows*), el número total de bacterias en la colonia (*total_cells*) y el porcentaje que supone la relación entre ambas cantidades (*rate*) cada 10min de simulación aproximadamente.

Para la primera prueba se ha añadido una probabilidad $P_{represor}(output0/input0) = 0{,}1$ y por tanto $P_{represor}(output1/input0) = 0{,}9$. Con estas probabilidades se pretende que la proteína GFP se active en el 10 % de las bacterias de la colonia debido al ruido. En los resultados a continuación, puede observarse que se mantiene este porcentaje de bacterias activadas por ruido durante toda la simulación, por tanto los resultados obtenidos son bastante precisos con el resultado esperado.

Resultados con ruido 0.1:

```
n_glows: 2, total_cells: 77, rate:2.5974%, time: 10.1
n_glows: 4, total_cells: 100, rate:4%, time: 20.2
n_glows: 7, total_cells: 155, rate:4.51613%, time: 30.3001
n_glows: 14, total_cells: 202, rate:6.93069%, time: 40.4
n_glows: 27, total_cells: 311, rate:8.68167%, time: 50.4998
n_glows: 45, total_cells: 419, rate:10.7399%, time: 60.5996
n_glows: 62, total_cells: 620, rate:10%, time: 70.6995
n_glows: 84, total_cells: 846, rate:9.92908%, time: 80.7993
n_glows: 113, total_cells: 1259, rate:8.97538%, time: 90.8992
n_glows: 178, total_cells: 1712, rate:10.3972%, time: 100.999
n_glows: 240, total_cells: 2542, rate:9.44138%, time: 111.099
n_glows: 352, total_cells: 3474, rate:10.1324%, time: 121.199
n_glows: 489, total_cells: 5098, rate:9.592%, time: 131.299
n_glows: 675, total_cells: 7023, rate:9.61128%, time: 141.399
n_glows: 965, total_cells: 10263, rate:9.40271%, time: 151.5
```

En las siguientes pruebas se ha añadido una probabilidad $P_{represor}(output0/input0) = 0{,}2$ y por tanto $P_{represor}(output1/input0) = 0{,}8$ y $P_{represor}(output0/input0) = 0{,}3$ y por tanto $P_{represor}(output1/input0) = 0{,}7$ pretendiendo obtener un 20 % y un 30 % de células activadas por ruido respectivamente. En ambos casos, se observa que se obtiene el resultado esperado durante la mayor parte de la simulación.

Resultados con ruido 0.2:

```
n_glows: 6, total_cells: 73, rate:8.21918%, time: 10.1
n_glows: 8, total_cells: 102, rate:7.84314%, time: 20.2
n_glows: 22, total_cells: 150, rate:14.6667%, time: 30.3001
n_glows: 36, total_cells: 209, rate:17.2249%, time: 40.4
n_glows: 53, total_cells: 302, rate:17.5497%, time: 50.4998
n_glows: 92, total_cells: 421, rate:21.8527%, time: 60.5996
n_glows: 125, total_cells: 610, rate:20.4918%, time: 70.6995
n_glows: 173, total_cells: 852, rate:20.3052%, time: 80.7993
n_glows: 244, total_cells: 1226, rate:19.9021%, time: 90.8992
n_glows: 348, total_cells: 1729, rate:20.1272%, time: 100.999
n_glows: 483, total_cells: 2475, rate:19.5152%, time: 111.099
n_glows: 678, total_cells: 3477, rate:19.4996%, time: 121.199
n_glows: 953, total_cells: 5001, rate:19.0562%, time: 131.299
n_glows: 1368, total_cells: 7019, rate:19.49%, time: 141.399
n_glows: 1875, total_cells: 10063, rate:18.6326%, time: 151.5
```

Resultados con ruido 0.3:

```
n_glows: 7, total_cells: 69, rate:10.1449%, time: 10.1
n_glows: 14, total_cells: 99, rate:14.1414%, time: 20.2
n_glows: 30, total_cells: 144, rate:20.8333%, time: 30.3001
n_glows: 46, total_cells: 197, rate:23.3503%, time: 40.4
n_glows: 70, total_cells: 287, rate:24.3902%, time: 50.4998
n_glows: 113, total_cells: 402, rate:28.109%, time: 60.5996
n_glows: 156, total_cells: 581, rate:26.8503%, time: 70.6995
n_glows: 230, total_cells: 811, rate:28.3601%, time: 80.7993
n_glows: 337, total_cells: 1166, rate:28.9022%, time: 90.8992
n_glows: 493, total_cells: 1642, rate:30.0244%, time: 100.999
n_glows: 717, total_cells: 2348, rate:30.5366%, time: 111.099
n_glows: 1007, total_cells: 3311, rate:30.4138%, time: 121.199
n_glows: 1461, total_cells: 4717, rate:30.9731%, time: 131.299
n_glows: 2031, total_cells: 6671, rate:30.4452%, time: 141.399
n_glows: 2890, total_cells: 9501, rate:30.4179%, time: 151.5
```

En todas las pruebas se consiguen resultados muy aproximados al valor insertado. Por lo tanto queda comprobado que el ruido funciona correctamente.

6.3. Supervivencia de plásmidos

Uno de los objetivos de este módulo genético es la posibilidad de ser usado para predecir las características que debe cumplir un circuito genético para realizar una acción concreta. En este caso, se requiere conocer cual debe ser la minima tasa de conjugación que debe de tener un plásmido para asegurar su supervivencia en la colonia. Además, las células que contengan este plásmido, sufren una carga metabólica al llevar insertado este plásmido que las hace disminuir su tasa de crecimiento en un 15 %.

El código utilizado para obtener los siguientes resultados puede encontrarse en el Apéndice D. En él se definen 2 plásmidos. El *plásmido1* es del cual se necesita conocer la tasa mínima de conjugación. Este plásmido contiene a los genes 1 y 2, ambos regidos por promotores constitutivos. Si una célula contiene este plásmido sufre una carga metabólica del 15 %, haciendo que crezca un 15 % más lenta. Por otro lado, el *plásmido2* contiene únicamente al *gen3* controlado por un promotor constitutivo. Este segundo plásmido se utiliza únicamente para obtener datos sobre la cantidad de bacterias Receptoras, Donantes o Transconjugadas que hay en la colonia.

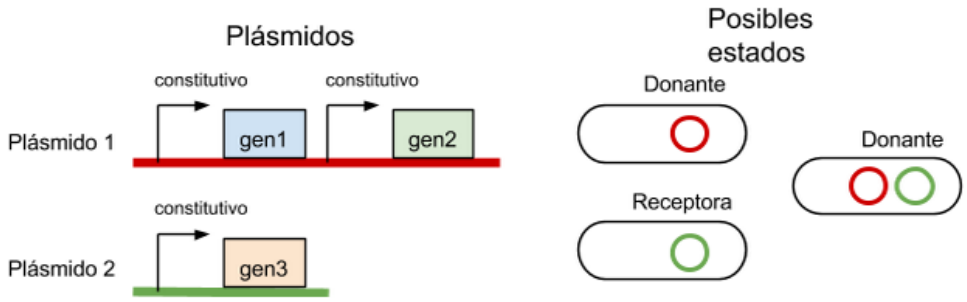


Figura 6.13: Diseño del circuito survival

Si una célula tiene la proteína sintetizada por el gen1 en estado *on*, ésta tendrá la capacidad de conjugar el plásmido1 con una tasa de conjugación indicada, la cual se necesita conocer. Si además, esta célula tiene a la proteína2 en estado *on*, la célula brillará con luz roja. Por tanto, en la simulación, las células rojas serán las que contienen el plásmido1 y las células negras las que únicamente contienen el plásmido2 (fig. 6.14).

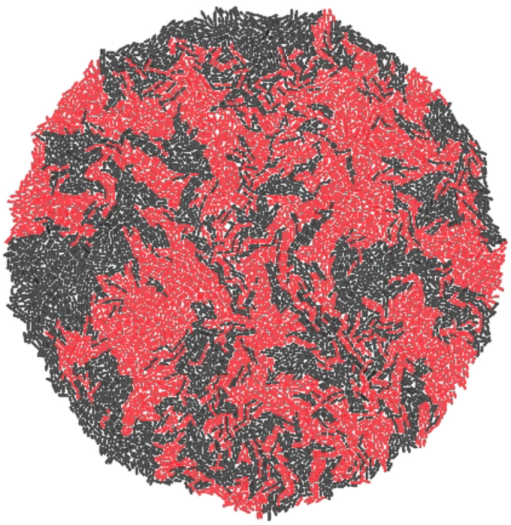


Figura 6.14: Ejemplo de simulación survival

Los datos obtenidos indican el número de células infectadas por el plásmido1 (*Infected*), el número de células que no están infectadas (*Uninfected*) y la diferencia entre

ambos valores (*Gap*) cada 10min aproximadamente. Se pretende conseguir una tasa de conjugación que consiga una diferencia lo más próxima a 0, ya que eso supondría que hay la misma cantidad de células de cada tipo, asegurando que a la larga el plásmido sobrevivirá. Si hay más cantidad de bacterias no infectadas, a la larga, es posible que esa diferencia sea muy grande como para que el plásmido sobreviva.Tras varios intentos, se ha obtenido como óptima tasa de conjugación 0.21 conjugaciones por ciclo de vida. Para demostrar que el valor óptimo ronda la tasa 0.2, se muestran también los resultados obtenidos con tasa 0.1 y tasa 0.3.

Resultados con tasa de conjugación 0.1:

```
Uninfected: 76, Infected: 68, Gap: 8, Time: 10.02
Uninfected: 98, Infected: 96, Gap: 2, Time: 20.12
Uninfected: 140, Infected: 129, Gap: 11, Time: 30.22
Uninfected: 188, Infected: 180, Gap: 8, Time: 40.32
Uninfected: 278, Infected: 238, Gap: 40, Time: 50.42
Uninfected: 366, Infected: 342, Gap: 24, Time: 60.52
Uninfected: 550, Infected: 455, Gap: 95, Time: 70.62
Uninfected: 733, Infected: 620, Gap: 113, Time: 80.72
Uninfected: 1073, Infected: 855, Gap: 218, Time: 90.82
Uninfected: 1467, Infected: 1133, Gap: 334, Time: 100.92
Uninfected: 2129, Infected: 1578, Gap: 551, Time: 111.02
Uninfected: 2925, Infected: 2092, Gap: 833, Time: 121.12
Uninfected: 4258, Infected: 2889, Gap: 1369, Time: 131.22
Uninfected: 5836, Infected: 3897, Gap: 1939, Time: 141.32
Uninfected: 8482, Infected: 5293, Gap: 3189, Time: 151.42
```

Resultados con tasa de conjugación 0.3:

```
Uninfected: 70, Infected: 76, Gap: −6, Time: 10.02
Uninfected: 87, Infected: 111, Gap: −24, Time: 20.12
Uninfected: 111, Infected: 168, Gap: −57, Time: 30.22
Uninfected: 148, Infected: 227, Gap: −79, Time: 40.32
Uninfected: 194, Infected: 319, Gap: −125, Time: 50.42
Uninfected: 269, Infected: 445, Gap: −176, Time: 60.52
Uninfected: 359, Infected: 592, Gap: −233, Time: 70.62
Uninfected: 509, Infected: 852, Gap: −343, Time: 80.72
Uninfected: 687, Infected: 1110, Gap: −423, Time: 90.82
Uninfected: 980, Infected: 1590, Gap: −610, Time: 100.92
Uninfected: 1339, Infected: 2074, Gap: −735, Time: 111.02
Uninfected: 1884, Infected: 2912, Gap: −1028, Time: 121.12
Uninfected: 2603, Infected: 3953, Gap: −1350, Time: 131.22
Uninfected: 3655, Infected: 5319, Gap: −1664, Time: 141.32
Uninfected: 5044, Infected: 7473, Gap: −2429, Time: 151.42
```

En estos resultados se observa que con una tasa de conjugación muy baja (0.1), a los

150min hay una diferencia de aproximadamente 3000 células entre las que no están infectadas y las que sí lo están. Lógicamente, hay un mayor número de células no infectadas ya que las células infectadas no poseen una tasa de conjugación suficiente para transmitir el plásmido a una mayor cantidad de células. Además, las células infectadas, debido a la carga metabólica, crecen más lentamente por lo que el número de células no se multiplica a la misma velocidad que las no infectadas.

En cuanto a los resultados con un tasa de conjugación de 0.3, se observa que en 150min existe una diferencia de unas 2500 bacterias entre las que tienen y el plásmido1 y las que no lo tienen. En este caso hay una mayor cantidad de células infectadas, por lo tanto, es una tasa demasiado alta. Esta tasa asegura la supervivencia del plásmido, pero no es la mínima.

La mínima tasa de conjugación que asegura la supervivencia del plásmido debe de estar contenida en el intervalo (0.1,0.3). Tras varios intentos con valores comprendidos en ese intervalo se concluye con que la mínima tasa de conjugación necesaria en este experimento con una carga metabólica del 15 % en las bacterias infectadas es de 0.21. Los resultados obtenidos con esta tasa pueden consultarse a continuación.

Resultados con tasa de conjugación 0.21:

```
Uninfected: 77, Infected: 80, Gap: -3, Time: 10.02
Uninfected: 94, Infected: 101, Gap: -7, Time: 20.12
Uninfected: 128, Infected: 165, Gap: -37, Time: 30.22
Uninfected: 167, Infected: 212, Gap: -45, Time: 40.32
Uninfected: 243, Infected: 290, Gap: -47, Time: 50.42
Uninfected: 320, Infected: 418, Gap: -98, Time: 60.52
Uninfected: 464, Infected: 523, Gap: -59, Time: 70.62
Uninfected: 619, Infected: 801, Gap: -182, Time: 80.72
Uninfected: 891, Infected: 992, Gap: -101, Time: 90.82
Uninfected: 1212, Infected: 1448, Gap: -236, Time: 100.92
Uninfected: 1738, Infected: 1890, Gap: -152, Time: 111.02
Uninfected: 2364, Infected: 2626, Gap: -262, Time: 121.12
Uninfected: 3407, Infected: 3638, Gap: -231, Time: 131.22
Uninfected: 4651, Infected: 4761, Gap: -110, Time: 141.32
Uninfected: 6618, Infected: 6925, Gap: -207, Time: 151.42
```

La diferencia entre el número de bacterias de cada tipo es de aproximadamente 200

bacterias en 150min. Teniendo en cuenta que la colonia tiene aproximadamente 13000 células totales, 200 bacterias de diferencia puede considerarse una cantidad despreciable. La diferencia entre ambas varía entre valores positivos y negativos debido a las divisiones de las células, ya que si se divide más cantidad de células infectadas, por ejemplo, se obtendrá un mayor número de ellas que de las no infectadas. Por lo tanto, la tasa de conjugación 0.21 asegura una cantidad similar de células de cada tipo, por lo tanto el plásmido asegura su supervivencia.

Capítulo 7

Conclusión

En este trabajo, se han presentado algunos aspectos fundamentales sobre las dinámicas moleculares. Estos organismos sencillos resultan ser sistemas enormemente complejos cuando se analizan las partes que los componen. La maquinaria celular es el resultado de un complejo entramado de interacciones entre diversos tipos de moléculas biológicas. Estas reacciones, por complicadas que sean, se pueden caracterizar por unas pocas leyes básicas. La aplicación de estas leyes básicas permite el desarrollo de modelos matemáticos capaces de describir redes celulares complejas, pero difíciles de interpretar intuitivamente.

En esta memoria se ha descrito un posible modelo booleano que caracteriza la regulación genética bacteriana mediante el uso de probabilidades y retrasos en las activaciones y degradaciones de las proteínas. Se ha optado por un diseño booleano debido a que es el más sencillo y, por tanto, menos costoso computacionalmente, lo que permite ejecutar el módulo genético individualmente en cada bacteria en colonias de al menos 10^5 individuos. Las probabilidades proporcionan estocasticidad a la simulación. Añadiendo a cada proceso una probabilidad de acierto, se crea un sistema no determinista que maneja las posibles mutaciones o fallos genéticos que pueden tener las células reales. Los retrasos en los tiempos reducen alguna de las restricciones del modelo booleano, simulando un tiempo de espera desde que se dan las condiciones para que un gen cambie de estado hasta que finalmente cambia de estado.

Además se ha modificado el lenguaje *gro* para permitir el diseño de circuitos genéticos

sintéticos. Estos circuitos son totalmente parametrizables. Cada uno de ellos, puede estar formado por uno o varios operones y regular varios genes. En los operones, están disponibles las puestas lógicas *yes, and* y *or* para manejar las entradas que activan el promotor.

En base a los resultados obtenidos al representar el circuito *Repressilator*, el simulador GRO permite representar una aproximación a las dinámicas de regulación genética. Dado que es un modelo booleano, no es posible recrear con tanta precisión este proceso como con EDOs, pero se obtienen resultados aproximados a los obtenidos en el laboratorio. Además, gracias a su sencillez, se consigue simular el comportamiento de bacterias con estos circuitos en colonias de 10^5 individuos.

Además, los resultados del experimento *Survival* han demostrado que el nuevo módulo permite estudiar dinámicas desconocidas diseñando modelos de experimentos. En este caso, se ha conseguido obtener un parámetro que debe poseer un plásmido para conseguir sobrevivir en una colonia de bacterias.

Por tanto, se ha creado una nueva herramienta que permite la simulación de colonias bacterianas para representar el resultado que causan los circuitos genéticos en ellas. Se ha demostrado que el modelo reproduce, en la medida de lo posible con un modelo booleano, el comportamiento de bacterias reales con el mismo circuito genético, por lo tanto se ha validado el comportamiento de este nuevo módulo, permitiendo su uso en el diseño de nuevos circuitos.

Bibliografía

[1] Alvarez, J. (1991). *Mapa bacteriológico de bacilos gram negativo* (Doctoral dissertation, Tesis de doctorado] Madrid, España: Universidad de Complutense, Facultad de medicina).

[2] Anderson, J. C., Voigt, C. A., & Arkin, A. P. (2007). *Environmental signal integration by a modular AND gate.* Molecular systems biology, 3(1), 133.

[3] Andrianantoandro, E., Basu, S., Karig, D. K., & Weiss, R. (2006). *Synthetic biology: new engineering rules for an emerging discipline.* Molecular systems biology, 2(1).

[4] Angel Goñi-Moreno and Martyn Amos.(2009).*DiSCUS: A Simulation Platform for Conjugation Computing.*

[5] Antonio P. García. Tesis del máster en Inteligencia Artificial. (2011). *A first approach to individual- based modeling of the bacterial conjugation dynamics.*

[6] Arteaga, J.P. Tesis del máster en Inteligencia Artificial. (2012). *Enhancing iDynoMICS framework to simulate Rod-shape bacterial colonies growth.*

[7] Aurora S. Arroyo García. Tesis del máster en Inteligencia Artificial. (2013).*Enhancing iDynoMICS framework with a plasmid conjugation module.*

[8] Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., ... & Densmore, D. (2011). *Eugene,a domain specific language for specifying and constraining synthetic biological parts, devices, and systems.* PloS one, 6(4), e18882.

[9] Bilitchenko, L., Liu, A., & Densmore, D. (2011). *The Eugene Language for Synthetic Biology.* Methods in enzymology, 498, 153.

[10] Cai, Y. (2010). *GenoCAD: linguistic approaches to synthetic biology* (Doctoral dissertation, Virginia Polytechnic Institute and State University).

[11] Chen, T., He, H. L., & Church, G. M. (1999, January). *Modeling gene expression with differential equations.* In Pacific symposium on biocomputing (Vol. 4, No. 29, p. 4).

[12] Czar, M. J., Cai, Y., & Peccoud, J. (2009). *Writing DNA with GenoCAD. Nucleic acids research*, 37(suppl 2), W40-W47.

[13] Dupuy, L., Mackenzie, J., Rudge, T., & Haseloff, J. (2008). *A system for modelling cell?cell interactions during plant morphogenesis.* Annals of botany, 101(8), 1255-1265.

[14] D'haeseleer, P., Wen, X., Fuhrman, S., & Somogyi, R. (1999, January). *Linear modeling of mRNA expression levels during CNS development and injury.* In Pacific symposium on biocomputing (Vol. 4, No. 1, pp. 41-52).

[15] Edwards A. L.& Batey R. T. (2010) *Riboswitches: A common RNA regulatory element.* Nature Education, 3(9):9,

[16] Elowitz, M. B., & Leibler, S. (2000). *A synthetic oscillatory network of transcriptional regulators.* Nature, 403(6767), 335-338.

[17] Gardner, T. S., Cantor, C. R., & Collins, J. J. (2000). *Construction of a genetic toggle switch in Escherichia coli.* Nature, 403(6767), 339-342.

[18] Gardner RG, et al. (2000) *Endoplasmic reticulum degradation requires lumen to cytosol signaling. Transmembrane control of Hrd1p by Hrd3p.* J Cell Biol 151(1):69-82

[19] Gorochowski TE, Matyjaszkiewicz A, Todd T, Oak N, Kowalska K, et al. (2012). *BSim: An Agent- Based Tool for Modeling Bacterial Populations in Systems and Synthetic Biology.* PLoS ONE 7(8): e42790. doi:10.1371/journal.pone.0042790

[20] Guet, C. C., Elowitz, M. B., Hsing, W., & Leibler, S. (2002). *Combinatorial synthesis of genetic networks.* Science, 296(5572), 1466-1470.

[21] Hasty, J., McMillen, D., & Collins, J. J. (2002). *Engineered gene circuits.* Nature, 420(6912), 224-230.

[22] Hatzimanikatis, V. (1999). *Nonlinear metabolic control analysis.* Metabolic engineering, 1(1), 75-87.

[23] Jacob, F., & Monod, J. (1961). *Genetic regulatory mechanisms in the synthesis of proteins.* Journal of molecular biology, 3(3), 318-356.

[24] Jan-Ulrich Kreft, Ginger Booth and Julian W. T. Wimpenny. (1998).*BacSim, a simulator for individual-based modelling of bacterial colony growth.*doi:10.1099/00221287-144-12- 3275Microbiology December 1998 vol. 144 no. 123275-3287

[25] Kauffman, S. (1969). *Homeostasis and differentiation in random genetic control networks.* Nature, 224, 177-178.

[26] Lardon, L. A., Merkey, B. V., Martins, S., Dötsch, A., Picioreanu, C., Kreft, J. U., & Smets, B. F. (2011). *iDynoMiCS: next?generation individual?based modelling of biofilms.* Environmental Microbiology, 13(9), 2416-2434.

[27] Levskaya, A., Chevalier, A. A., Tabor, J. J., Simpson, Z. B., Lavery, L. A., Levy, M., ... & Voigt, C. A. (2005). *Synthetic biology: engineering Escherichia coli to see light.* Nature, 438(7067), 441-442.

[28] Maki, Y., Tominaga, D., Okamoto, M., Watanabe, S., & Eguchi, Y. (2001, January). *Development of a system for the inference of large scale genetic networks.* In Pacific Symposium on Biocomputing (Vol. 6, pp. 446-458).

[29] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., & Alon, U. (2002). *Network motifs: simple building blocks of complex networks.* Science, 298(5594), 824-827.

[30] Pardilla Mata, D. (2011). *Estudio de protocolos de comunicación bacterianos: conjugación y quorum sensing* (Doctoral dissertation, Informatica).

[31] Park et al, (2003) *Rewiring MAP Kinase Pathways Using Alternative Scaffold Assembly Mechanisms* Science 299, 1061. doi: 10.1126/science.1076979

[32] Pedersen, M., & Phillips, A. (2009). *Towards programming languages for genetic engineering of living cells. Journal of the Royal Society Interface*,6(Suppl 4), S437-S450.

[33] Rodríguez-Patón Aradas, A. (1999). *Variantes de la concatenación en computación con ADN* (Doctoral dissertation, Informatica).

[34] Rudge, T. J., Steiner, P. J., Phillips, A., & Haseloff, J. (2012). *Computational modeling of synthetic microbial biofilms.* ACS Synthetic Biology, 1(8), 345-352.

[35] Santos, J. C. R. *Sistemas Regulatorios de la Expresión Génica.*

[36] Serrano Sebastián, I. (2014). *Programación y simulación de la comunicación intercelular mediante conjugación en el simulador multicelular GRO.*

[37] Seunghee S. Jang, Kevin T. Oishi, Robert G. Egbert, and Eric Klavins. (2012). *Specification and Simulation of Synthetic Multicelled Behaviors.* doi:10.1021/sb300034m — ACS Synth.

[38] Smolen, P., Baxter, D. A., & Byrne, J. H. (2000). *Modeling transcriptional control in gene networks: methods, recent results, and future directions.* Bulletin of mathematical biology, 62(2), 247-292.

[39] Smolen, P., Baxter, D. A., & Byrne, J. H. (2000). *Mathematical modeling of gene networks.* Neuron, 26(3), 567-580.

[40] Tkacik, G., Gregor, T., & Bialek, W. (2008). *The role of input noise in transcriptional regulation.* PLoS One, 3(7), e2774.

[41] Umesh, P., Naveen, F., Rao, C. U. M., & Nair, A. S. (2010). *Programming languages for synthetic biology. Systems and synthetic biology,* 4(4), 265-269.

[42] Voigt, C. A. (2006). *Genetic parts to program bacteria.* Current opinion in biotechnology, 17(5), 548-557.

[43] Wahde, M., & Hertz, J. (2001). *Modeling genetic regulatory dynamics in neural development.* Journal of computational Biology, 8(4), 429-442.

[44] Weiss, R., Basu, S., Hooshangi, S., Kalmbach, A., Karig, D., Mehreja, R., & Netravali, I. (2003). *Genetic circuit building blocks for cellular computation, communications, and signal processing.* Natural Computing, 2(1), 47-84.

[45] Wilson, M. L., Hertzberg, R., Adam, L., & Peccoud, J. (2011). *A Step-by-Step Introduction to Rule-Based Design of Synthetic Genetic Constructs Using GenoCAD.* Methods in enzymology, 498, 173.

Apéndice A

Diseño general de un circuito genético en gro

```
include gro

set ( "dt", 0.05 );
set ( "population_max", 20000 );

program p() := {

  skip();

};

set ("num_plasmids",2);
set ("num_proteins",4);
set ("num_riboswitchs",4);
set ("num_molecules",4);

degradation_times({10.0,5.0,5.0,2.0},{1.2,2.3,1.4,1.2},
                  {2.0,3.0,2.0,4.0},{0.2,0.3,0.4,0.2});

RNA_matrix({true,false,false,true,
            false,false,true,false,
            false,true,false,true,
            false,false,false,true});
```

```
molecules_matrix({0,-1,1,0,
                  0,0,1,0,
                  1,0,0,0,
                  0,-1,0,0});

operon ({false,true,false,false}, {false,false,false,false},
        {0,0,1,0}, {0,1,0,0}, false,
        {10.0,10.0,10.0,10.0}, {1.2,2.3,1.4,1.2},
        {2.0,3.0,2.0,4.0}, {0.2,0.3,0.4,0.2},
         0, {1,0,0,1}, {0,1,1,0});
operon ({false,true,false,false}, {false,false,false,false},
        {0,0,0,-1}, {0,0,0,0}, false,
        {10.0,10.0,10.0,10.0}, {1.2,2.3,1.4,1.2},
        {2.0,3.0,2.0,4.0}, {0.2,0.3,0.4,0.2},
         0, {1,0,0,1}, {0,1,1,0});
operon ({true,false,false,true}, {false,false,false,false},
        {1,0,1,0}, {0,0,0,1}, false,
        {20.0,20.0,20.0,20.0}, {1.2,2.3,1.4,1.2},
        {2.0,3.0,2.0,4.0}, {0.2,0.3,0.4,0.2},
         2, {1,0,0,1}, {0,1,1,0});
operon ({false,false,true,false},{false,false,false,false},
        {0,0,0,0}, {0,0,0,0}, true,
        {30.0,30.0,30.0,30.0}, {1.2,2.3,1.4,1.2},
        {2.0,3.0,2.0,4.0}, {0.2,0.3,0.4,0.2},
         0, {1,0,0,1}, {0,1,1,0});

plasmids_matrix ({true,true,true,true,
                  false,true,false,false});

set_molecule(1,1);

ecoli ( [ x := 0, y := 0], {true, false}, {false,false,false,false},
          {false,false,false,false},program p() );

action({false,true,false},"paint",{"100","0","0","0"});
action({false,true,false},"conjugate",{"1","20.0"});
action({false,false,true},"die",{"10"});
action({false,false,true},"paint",{"0","100","0","0"});
action({false,false,false},""paint",{"0","0","100","0"});
```

Apéndice B

Diseño del Repressilator en gro

```
include gro

set ( "dt", 0.1 );
set ( "population_max", 20000 );

/*The total number of plasmids present in the simulation.*/
set ( "num_plasmids", 2 );
/*The total number of proteins present in the simulation.*/
set ( "num_proteins", 4 );

/*Changing the display theme...*/
set_theme(dark_theme);

/*A time keeping variable.*/
t := 0;

/*A counter for frames to be recorded as pictures.*/
n := 0;

/*CIRCUIT DEFINITION:

Proteins: [TetR, LacI, cI, GFP]
Plasmid 1: LacI represses TetR
           TetR represses cI
           cI represses LacI
Plasmid 2: TetR represses GFP */
```

```
/*Degradation times for proteins*/
degradation_times({30.0170,32.2900,30.0170,33.8000}, {2.0,2.0,2.0,2.0},
                  {0.0,0.0,0.0,0.0},  {0.0,0.0,0.0,0.0});

/*TetR Operon*/
operon ({true,false,false,false}, {false,false,false,false},
        {0,-1,0,0}, {}, false,
        {32.2900,0.0,0.0,0.0}, {20.3,0.0,0.0,0.0},
        {0.0,0.0,0.0,0.0}, {0.0,0.0,0.0,0.0}, 0,
        {0.0,0.0,0.0,0.0}, {0.0038,0.9962,1,0});

/*LacI Operon*/
operon ({false,true,false,false}, {false,false,false,false},
        {0,0,-1,0}, {}, false,
        {0.0,30.0170,0.0,0.0}, {0.0,20.3,0.0,0.0},
        {0.0,0.0,0.0,0.0}, {0.0,0.0,0.0,0.0},
         0, {0.0,0.0,0.0,0.0},{0.0003,0.997,1,0});
/*cI Operon*/
operon ({false,false,true,false}, {false,false,false,false},
        {-1,0,0,0}, {}, false,
        {0.0,0.0,30.0170,0.0}, {0.0,0.0,20.3,0.0},
        {0.0,0.0,0.0,0.0}, {0.0,0.0,0.0,0.0},
         0, {0.0,0.0,0.0,0.0},{0.0088,0.9912,1,0});
/*GFP Operon*/
operon ({false,false,false,true}, {false,false,false,false},
        {-1,0,0,0}, {}, false,
        {0.0,0.0,0.0,30.0170}, {0.0,0.0,0.0,20.3},
        {0.0,0.0,0.0,0.0}, {0.0,0.0,0.0,0.0},
         0, {0.0,0.0,0.0,0.0},{0.0088,0.9912,1,0});

/*Operon-to-plasmid assignation matrix*/
plasmids_matrix ({true,true,true,false,
                  false,false,false,true});

action({false,false,false,false},"set_growth_rate",{"0.0011"});

action({false,false,false,true},"paint",{"100","0","0","0"});
action({true,false,false,false},"paint",{"0","0","0","0"});
action({false,true,false,false},"paint",{"0","0","0","0"});
```

```
program p() :=
{
        skip();
};

c_ecolis(30, 50, {true, true},
     {false,false,false,false}, {false,false,false,false}, program p());
```

Apéndice C

Test de ruido en gro

```
include gro

set ( "dt", 0.1 );
set ( "population_max", 2000000 );

program p() := {
  skip();

};

set ("num_plasmids",1);
set ("num_proteins",2);

t := 0;
ruido := 0.1;

degradation_times({3.0,1.0},{0.2,0.5},
                  {0.0,0.0},{0.0,0.0});

operon ({true,false}, {false,false}, {0,0}, {}, true,
        {1.0,0.0}, {0,0}, {0,0}, {0,0},
         0, {1.0,0.0,0.0,1.0},{0.0,1.0,1.0,0.0});
operon ({false,true}, {false,false}, {-1,0}, {}, false,
        {0.0,2.0}, {0,0}, {0,0}, {0,0},
         0, {1.0,0.0,0.0,1.0},{ruido,1.0,1-ruido,0.0});
```

```
plasmids_matrix ({true,true});

action({false,true},"paint",{"20","0","0","0"});
action({true,false},"paint",{"0","0","0","0"});

c_ecolis(50, 120, {true}, {true,false}, {true,false}, program p());
```

Apéndice D

Survival en gro

```
include gro

set ( "dt", 0.1 );
set ( "population_max", 2000000 );

T := 0;
Td := 0;
R := 0;
D := 0;

program p() := {

  rfp := 0;
  counted := false;
  count_conj := false;

  received_plasmid(1) & has_plasmid(2) & !count_conj:
  {
              T := T + 1;
              R := R - 1;
              count_conj := true;
  }

  has_plasmid(1):
  {
              set("ecoli_growth_rate", 0.0295); //15%
```

```
                rfp := rfp+50;
  }

  !has_plasmid(1) & has_plasmid(2) & !counted:
  {
                R := R + 1;
                counted := true;
  }

  has_plasmid(1) & !has_plasmid(2) & !counted:
  {
                D := D + 1;
                counted := true;
  }

  daughter & received_plasmid(1) & has_plasmid(2) & counted:
  {
                Td := Td + 1;
  }

  daughter & !has_plasmid(1) & has_plasmid(2) & counted:
  {
                R := R + 1;
  }

  daughter & has_plasmid(1) & counted:
  {
                D := D + 1;
  }
};

set ("num_plasmids",2);
set ("num_proteins",3);

t := 0;

degradation_times({0.0,0.0,0.0},{0.0,0.0,0.0},
                  {0.0,0.0,0.0},{0.0,0.0,0.0});

operon ({true,false,false}, {false,false,false}, {0,0,0,0}, {}, true,
```

```
        {10.0,0.0,0.0}, {0,0,0}, {0,0,0}, {0,0,0},
         0, {1.0,0.0,0.0,1.0},{0.0,1.0,1.0,0.0});
operon ({false,true,false}, {false,false,false}, {0,0,0,0}, {}, true,
        {0.0,1.0,0.0}, {0,0,0}, {0,0,0}, {0,0,0},
         0, {1.0,0.0,0.0,1.0},{0.0,1.0,1.0,0.0});
operon ({false,false,true}, {false,false,false}, {0,0,0,0}, {}, true,
        {0.0,0.0,1.0}, {0,0,0}, {0,0,0}, {0,0,0},
         0, {1.0,0.0,0.0,1.0},{0.0,1.0,1.0,0.0});

plasmids_matrix ({true,true,false,
                  false,false,true});

action({true,false,false},"conjugate",{"0","0.21"});
action({false,true,false},"d_paint",{"0","50","0","0"});

program main() :=
{

        print_t := 0;

        true:
        {
                t := t + dt;
                print_t := print_t + 1;
        }

        print_t > 1 & print_t=101:
        {
                print("D:", D, ", R:", R, ", T: ", T, ", Td:", Td, "\n");
                print("Uninfected: ", R, ", Infected: ", (D+T+Td), ",
                Gap: ", (R-D-T-Td), ", Time: ",t, "\n");
                print_t := 0;
        }

        c_ecolis(50, 120, {true, false}, {false,false,false},
                {false,false,false}, program p());
        c_ecolis(50, 120, {false, true}, {false,false,false},
                {false,false,false}, program p());
};
```